

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problems Mailbox.**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

#3
10-12-01

In re U.S. Patent Application of)
KAMINAGA et al.)
Application Number: To be assigned)
Filed: Concurrently herewith)
For: TAMPER-RESISTANT MODULAR)
MULTIPLICATION METHOD)

1c996 U.S. PTO
09/935654
08/24/01

Honorable Assistant Commissioner
for Patents
Washington, D.C. 20231

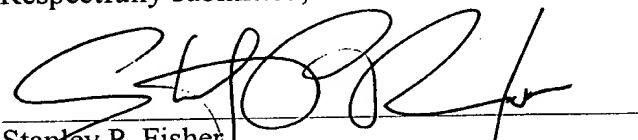
REQUEST FOR PRIORITY
UNDER 35 U.S.C. § 119
AND THE INTERNATIONAL CONVENTION

Sir:

In the matter of the above-captioned application for a United States patent, notice is hereby given that the Applicant claims the priority date of March 5, 2001, the filing date of the corresponding Japanese patent application 2001-060223.

The certified copy of corresponding Japanese patent application 2001-060223 is submitted herewith. Acknowledgment of receipt of the certified copy is respectfully requested in due course.

Respectfully submitted,


Stanley P. Fisher
Registration Number 24,344

REED SMITH HAZEL & THOMAS LLP
3110 Fairview Park Drive
Suite 1400
Falls Church, Virginia 22042
(703) 641-4200
August 24, 2001

JUAN CARLOS A. MARQUEZ
Registration No. 34,072

jc996 U.S. PTO

09/935654



08/24/01

PATENT OFFICE JAPANESE GOVERNMENT

This is to certify that the annexed is a true copy of the following application as filed with this office.

Date of Application : March 5, 2001
Application Number : Patent Application No. 060223 of 2001
Applicant (s) : Hitachi, Ltd.

Dated this 27th day of July, 2001

Kouzou OIKAWA
Commissioner,
Patent Office
Certificate No. 2001-3066032

日 本 国 特 許 庁
JAPAN PATENT OFFICE



別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application:

2001年 3月 5日

出 願 番 号

Application Number:

特願2001-060223

出 願 人

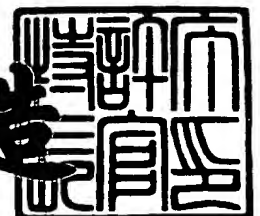
Applicant(s):

株式会社日立製作所

2001年 7月27日

特許庁長官
Commissioner,
Japan Patent Office

及川耕造



出証番号 出証特2001-3066032

【書類名】 特許願

【整理番号】 NT00P1115

【提出日】 平成13年 3月 5日

【あて先】 特許庁長官 殿

【国際特許分類】 G06K 19/00

【発明者】

 【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目280番地 株式会社日立製作所 中央研究所内

 【氏名】 神永 正博

【発明者】

 【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目280番地 株式会社日立製作所 中央研究所内

 【氏名】 遠藤 隆

【発明者】

 【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目280番地 株式会社日立製作所 中央研究所内

 【氏名】 渡邊 高志

【発明者】

 【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目280番地 株式会社日立製作所 中央研究所内

 【氏名】 大木 優

【特許出願人】

 【識別番号】 000005108

 【氏名又は名称】 株式会社日立製作所

【代理人】

 【識別番号】 100068504

 【弁理士】

 【氏名又は名称】 小川 勝男

 【電話番号】 03-3661-0071

【選任した代理人】

【識別番号】 100086656

【弁理士】

【氏名又は名称】 田中 恭助

【電話番号】 03-3661-0071

【選任した代理人】

【識別番号】 100094352

【弁理士】

【氏名又は名称】 佐々木 孝

【電話番号】 03-3661-0071

【手数料の表示】

【予納台帳番号】 081423

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 耐タンパーモジュラ演算処理方法

【特許請求の範囲】

【請求項 1】

情報処理装置を利用して、暗号処理中に出現する剰余乗算である $A * B * R^{(-1) \bmod N}$

(-1) mod N を計算する方法であって、その方法は、

(1) $S_1 = A * B * R^{(-1) \bmod N}$ を計算するステップ、

(2) (1) の代わりに $S_2 = \{sN + A * (-1)^f\} * \{tN + B * (-1)^g\} * R^{(-1) \bmod N}$ (s, t, f, g は少なくとも1つは0でない整数であり、 f, g は0又は1) を計算するステップ、

(3) 上記(1)と(2)のいずれかを適宜選択するステップ、

(4) 上記(1)(2)(3)を適宜繰り返し、最後に上記(1)を選択したときには計算結果 S_1 について $T_1 = S_1 * R^{(-1) \bmod N}$ を計算して T_1 を出力し、上記(2)を選択したときには計算結果 S_2 について $T_2 = S_2 * R^{(-1) \bmod N}$ を計算して $N - T_2$ を出力するステップ、および

(5) T_1 と $N - T_2$ を、暗号処理の剰余乗算 $A * B * R^{(-1) \bmod N}$ の計算結果として用いるステップを有することを特徴とする耐タンパーモジュラ演算処理方法。

【請求項 2】

前記(3)の適宜選択するとは、乱数を用いていずれかを選択することを特徴とする請求項1記載の耐タンパーモジュラ演算処理方法。

【請求項 3】

前記(s, t, f, g)は、(0, 1, 0, 1)であることを特徴とする請求項1記載の耐タンパーモジュラ演算処理方法。

【請求項 4】

前記(s, t, f, g)は、(1, 0, 1, 0)であることを特徴とする請求項1記載の耐タンパーモジュラ演算処理方法。

【請求項 5】

情報処理装置を利用して、暗号処理中に出現する剰余乗算である $A * B \bmod$

d_p (p は素数) を計算する方法であって、その方法は、

- (1) $S = A * B \bmod p$ を計算するステップ、
- (2) (1) の代わりに $S = \{s_p + A * (-1)^f\} * \{t_p + B * (-1)^g\} \bmod p$ (s, t, f, g は少なくとも 1 つは 0 でない整数であり、 f, g は 0 または 1 で $f + g$ は偶数) を計算するステップ、
- (3) 上記 (1) と (2) のいずれかを適宜選択するステップ、および
- (4) 上記 (1) と (2) のうち選択された方の計算結果 S を暗号処理の剰余乗算 $A * B \bmod p$ の計算結果として用いるステップを有することを特徴とする耐タンパーモジュラ演算処理方法。

【請求項 6】

前記 (s, t, f, g) は、(1, 1, 1, 1) であることを特徴とする請求項 5 記載の耐タンパーモジュラ演算処理方法。

【請求項 7】

- 前記 (2) のステップの $f + g$ は奇数であって、前記 (4) の代わりに、
- (4) 前記 (1) を選択したときは計算結果 S そのままとし、前記 (2) を選択したときには S の代わりに $p - S$ を計算結果とするステップ、および
- (5) 前記 S と $p - S$ を、暗号処理の剰余乗算 $A * B \bmod p$ の計算結果として用いるステップを有することを特徴とする請求項 5 記載の耐タンパーモジュラ演算処理方法。

【請求項 8】

前記 (s, t, f, g) は、(0, 1, 0, 1) であることを特徴とする請求項 7 記載の耐タンパーモジュラ演算処理方法。

【請求項 9】

情報処理装置を利用して、暗号処理中に出現する剰余乗算である $A(x) * B(x) \bmod \Phi(x)$ ($\Phi(x)$ は x の既約多項式であり、 $A(x) * B(x)$ の係数の演算は 3 以上の素数 p を法として行う) を計算する方法であって、その方法は、

- (1) $S(x) = A(x) * B(x) \bmod \Phi(x)$ を計算するステップ、
- (2) (1) の代わりに $S(x) = \{s \Phi(x) + A(x) * (-1)^f\} * \{t$

$\Phi(x) + B(x) * (-1)^g \} \bmod \Phi(x)$ (s, t, f, g は少なくとも1つは0でない整数であり、 f, g は0又は1で $f + g$ は偶数) を計算するステップ、

(3) 上記(1)と(2)のいずれかを適宜選択するステップ、および

(4) 上記(1)と(2)のうち選択された方の計算結果 $S(x)$ を、暗号処理の剰余乗算 $A(x) * B(x) \bmod \Phi(x)$ の計算結果として用いるステップを有することを特徴とする耐タンパーモジュラ演算処理方法。

【請求項10】

前記(s, t, f, g)は、(1, 1, 1, 1)であることを特徴とする請求項9記載の耐タンパーモジュラ演算処理方法。

【請求項11】

前記(2)のステップの $f + g$ は奇数であって、前記(4)の代わりに、

(4) 前記(1)を選択したときは計算結果 $S(x)$ そのままとし、上記(2)を選択したときには $S(x)$ の代わりに $\Phi(x) - S(x)$ を計算結果とするステップ、および

(5) 前記 $S(x)$ と $\Phi(x) - S(x)$ を、暗号処理の剰余乗算 $A(x) * B(x) \bmod \Phi(x)$ の計算結果として用いるステップを有することを特徴とする請求項9記載の耐タンパーモジュラ演算処理方法。

【請求項12】

前記(s, t, f, g)は、(0, 1, 0, 1)であることを特徴とする請求項11記載の耐タンパーモジュラ演算処理方法。

【請求項13】

前記 $A(x) * B(x)$ の係数の演算は素数2を法として行うものであり、前記(2)の(f, g)は(0, 0)であることを特徴とする請求項9記載の耐タンパーモジュラ演算処理方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、高いセキュリティを持つICカードなどの耐タンパー暗号処理方法

に関するものである。

【 0 0 0 2 】

【従来の技術】

ＩＣカードは、勝手に書き換えることが許されないような個人情報の保持や、秘密情報である暗号鍵を用いたデータの暗号化や暗号文の復号化を行う装置である。ＩＣカード自体は電源を持っておらず、ＩＣカード用のリーダライタに差し込まれると、電源の供給を受け、動作可能となる。動作可能になると、リーダライタから送信されるコマンドを受信し、そのコマンドに従ってデータの転送等の処理を行う。ＩＣカードの一般的な解説は、オーム社出版電子情報通信学会編水沢順一著「ＩＣカード」などにある。

【 0 0 0 3 】

ＩＣカードの構成は、図１に示すように、カード１０１の上にＩＣカード用チップ１０２を搭載したものである。図に示すように、一般にＩＣカードは、ＩＳＯ 7 8 1 6 の規格に定められた位置に供給電圧端子Vcc、グランド端子GND、リセット端子RST、入出力端子I/Oおよびクロック端子CLKを持ち、これらの端子を通して、リーダライタから電源の供給やリーダライタとのデータの通信を行う(W.Rankl and Effing : SMARTCARD HANDBOOK, John Wiley & Sons, 1997, pp.41 参照)。

【 0 0 0 4 】

ＩＣカード用チップの構成は、基本的には通常のマイクロコンピュータと同じ構成である。その構成は、図２に示すように、中央処理装置(CPU)201、記憶装置204、入出力(I/O)ポート207、コ・プロセッサ202からなる(コ・プロセッサはない場合もある)。CPU201は、論理演算や算術演算などを行う装置であり、記憶装置204は、プログラムやデータを格納する装置である。入出力ポートは、リーダライタと通信を行う装置である。コ・プロセッサは、暗号処理そのもの、または暗号処理に必要な演算を高速に行う装置であり、例えばRSA暗号の剰余演算を行うための特別な演算装置や、DES暗号のラウンド処理を行う暗号装置などがある。ＩＣカード用プロセッサの中には、コ・プロセッサを持たないものも多くある。データバス203は、各装置を接続するバスである。

【0005】

記憶装置204は、ROM(Read Only Memory)やRAM(Random Access Memory)、EEPROM(Electrical Erasable Programmable Read Only Memory)などからなる。ROMは、変更できないメモリであり、主にプログラムを格納するメモリである。RAMは自由に書き換えができるメモリであるが、電源の供給が中断されると、記憶している内容は消滅する。ICカードがリーダライタから抜かれると電源の供給が中断されるため、RAMの内容は、保持されなくなる。EEPROMは、電源の供給が中断されてもその内容を保持することができるメモリである。書き換える必要があり、ICカードがリーダライタから抜かれても、保持するデータを格納するために使われる。例えば、プリペイドカードでのプリペイドの度数などは、使用するたびに書き換えられ、かつリーダライタが抜かれてもデータを保持する必要があるため、EEPROMで保持される。

【0006】

ICカードは、プログラムや重要な情報がICカード用チップの中に密閉されているため、重要な情報を格納したり、カードの中で暗号処理を行うために用いられる。従来、ICカードでの暗号を解読する難しさは、暗号アルゴリズムの解読の困難さと同じと考えられていた。しかし、ICカードが暗号処理を行っている時の消費電流を観測し、解析することにより、暗号アルゴリズムの解読より容易に暗号処理の内容や暗号鍵が推定される可能性が示唆されている。消費電流は、リーダライタから供給されている電流を測定することにより観測することができ、この攻撃法の詳細は、John Wiley & sons社 W.Rankl & W.Effing著「Smart Card Handbook」の8.5.1.1 Passive protective mechanisms(263ページ)にこのような危険性が記載されている。

【0007】

ICカード用チップを構成しているCMOSは、出力状態が1から0あるいは0から1に変わった時に電流を消費する。特に、データバス203においては、バスドライバの電流や、配線および配線に接続されているトランジスタの静電容量のため、バスの値が1から0あるいは0から1に変わると、大きな電流が流れる。そのため、消費電流を観測すれば、ICカード用チップの中で、何が動作し

ているか分かる可能性がある。

【0008】

図3は、ICカード用チップの1サイクルでの消費電流の波形を示したものである。処理しているデータに依存して、電流波形が301や302のように異なる。このような差は、バス203を流れるデータや中央演算装置201で処理しているデータに依存して生じる。

【0009】

コ・プロセッサ202は、CPUと並列に、例えば、512ビットの剰余演算を行うことができる。そのため、CPUの消費電流とは異なった消費電流波形の長時間の観測が可能である。その特徴的な波形を観測することにより、コ・プロセッサの動作回数を容易に測定することができる。コ・プロセッサの動作回数が暗号鍵と何らかの関係があるならば、コ・プロセッサの動作回数から暗号鍵を推定できる可能性がある。

【0010】

また、コ・プロセッサでの演算内容が、暗号鍵に依存した偏りがあると、その偏りが消費電流から求められ、暗号鍵が推定される可能性がある。例えば、乗算剰余演算の際に発生するオーバーフロー処理は、多くの場合、オーバーフロー処理特有の電流が発生する。オーバーフロー処理をするかしないかで、処理時間が異なる場合もある。

【0011】

CPUでも同様の事情が存在する。暗号鍵のビット値は決まっているため、処理するデータを変更して、消費電流を観測することにより、暗号鍵のビット値の影響が観測できる可能性がある。これらの消費電流の波形を統計的に処理することにより、暗号鍵を推定できる可能性がある。

【0012】

【発明が解決しようとしている課題】

本発明の課題は、ICカード用チップでのデータ処理と消費電流との関連性を減らすことである。消費電流とチップの処理との関連性が減れば、観測した消費電流の波形からICカードチップ内での処理や暗号鍵の推測が困難になる。本発

明の着眼点は、ICカードチップでの乗算剰余処理 $AB \bmod N$ またはモンゴメリ法による乗算剰余処理 $ABR^{(-1)} \bmod N$ において、乗数 B または被乗数 A を、そのモジュラス N を用いて、 $tN + B \cdot (-1)^g$ 、 $sN + A \cdot (-1)^f$ (s, t, f, g は整数。但し、 f, g は 0 又は 1) に置き換えて処理することにより、消費電流の波形や処理時間の偏りから、処理内容や暗号鍵の推測を困難にするものである。

【0013】

【課題を解決するための手段】

ICカードチップに代表される耐タンパー装置は、プログラムを格納するプログラム格納部、データを保存するデータ格納部を持つ記憶装置と、プログラムに従って所定の処理を実行しデータ処理を行う中央演算装置(CPU)を持ち、プログラムは、CPUに実行の指示を与える処理命令から構成される一つ以上のデータ処理手段からなる情報処理装置として捉えることができる。

【0014】

電子マネーに代表される高セキュリティのICカードでは、RSA暗号や、ガロア体 $GF(p^n)$ (p は素数、 n は正の整数) 上の楕円曲線暗号が用いられる。その際、乗算剰余演算 $A \cdot B \bmod N$ や、多項式の乗算剰余演算 $A(X) \cdot B(X) \bmod \Phi(X)$ の処理が必要となる ($A=B$ の場合もある)。以下、多項式の場合も考え方は同じなので、通常の乗算剰余処理の場合について説明する。多項式の場合の乗算剰余処理については、計算機上の実装が通常の乗算剰余処理の場合と若干異なるが、それは「発明の実施形態」にて詳細に説明する。

【0015】

本発明において、乗算剰余演算で処理しているデータとICカード用チップの消費電流の関連性を攪乱する一つの方法は、本来の演算処理に用いられるデータを直接用いずに、演算結果が大きく変わらないようなデータを用いて計算を行い、計算後に結果を修正することである。

【0016】

具体的には、乗算剰余処理 $A \cdot B \bmod N$ の計算を行う際、乗数 B 及び被乗数 A の両者を、それぞれ $tN + B \cdot (-1)^g$ 、 $sN + A \cdot (-1)^f$ (s, t, f, g は少なくとも1つは 0 でない整数。但し、 f, g は 0 又は 1) に置き換えて、

$$S = (sN + A*(-1)^f)*(tN + B*(-1)^g) \bmod N$$

を計算し、その後、 $f+g$ が奇数になる場合は、 $N-S$ を該乗算剰余演算結果とし、 $f+g$ が偶数となる場合は、 S を演算結果とすることである。

【0017】

上記の処理では、乗算剰余演算を行う度に本来の答になるように修正している。しかし、RSA暗号に代表されるべき乗剰余計算の場合は、毎回修正を施す必要はなく、最終的に修正を施せばよい。

【0018】

RSA暗号で用いられるべき乗剰余計算 $y^x \bmod N$ に対して用いられる乗算剰余処理を上記の置き換え、すなわち、乗算剰余処理 $A*B \bmod N$ の計算を行う際、乗数 B 及び被乗数 A の両者を、それぞれ $tN + B*(-1)^g$ 、 $sN + A*(-1)^f$ (s, t, f, g は整数。但し、 f, g は0又は1)に置き換えて、

$$S = (sN + A*(-1)^f)*(tN + B*(-1)^g) \bmod N$$

を計算し、最後に、必要なら N からべき乗剰余計算の値を引く修正を行えばよい。なぜこのようなことができるかは、「発明の実施形態」の中で述べる。

【0019】

上記のように処理することで、本来の乗算剰余演算処理で発生する電流とは異なるパターンの電流が観測されることになるため、本来のデータの場合に期待される波形を基に内部処理を推定することが困難になる。 s, t, f, g を外部から推定できない情報源を用いて変更すれば、さらに効果が高まり、特に、複数の波形を収集して平均をとるなどの統計処理を行う場合は、ランダムな波形を平均化することと同じことになり、波形の特徴は消滅するため、さらに効果は顕著である。

【0020】

本発明は、特にRSA暗号での、乗算剰余演算や、べき乗剰余計算、および楕円曲線暗号での定義体上での乗算や除算、ベースポイントのスカラー倍などの情報隠蔽に利用することができる。

【0021】

【発明の実施形態】

以下に示す実施例では、公開鍵暗号（非対称鍵暗号）の代表例であるRSA暗号、楕円曲線暗号を例にとる。RSA暗号については、岡本栄司著「暗号理論入門」（共立出版）や、A.J.Menezes、P.C. van Oorschot、S. A. Vanstone著 Handbook of Applied Cryptography、(CRC-Press)などに詳しく記載されている。楕円曲線暗号については、考案者の一人が書いたN. コブリツ著（櫻井幸一訳）「数論アルゴリズムと楕円暗号理論入門」（シュプリンガーフェアラーク東京）、楕円曲線上の演算については、I.H.シルバーマン・J.テイト著「楕円曲線論入門」（シュプリンガーフェアラーク東京）、又、群、環、体等の代数系については、松坂和夫著「代数系入門」（岩波書店）に詳しい説明がある。

【0022】

実施例を説明するに先立ち、背景となる数学的知識を整理しておく。一般に公開鍵暗号（非対称鍵暗号）においては、秘密鍵情報が公開鍵の中に含まれているが、公開鍵から秘密鍵情報を取り出すことが、計算時間の点で著しく現実性を欠くということ（計算量的安全性）を根拠にして暗号が構成されている。計算量的安全性を持つ問題の代表的なものとして、素因数分解と、群上の離散対数問題が挙げられる。前者を利用したものがRSA暗号であり、後者を楕円曲線上のモデル・ヴェイユ群に適用して利用しているものが楕円曲線暗号である。

【0023】

簡単にRSA暗号を説明する。RSA暗号では、大きな素数、例えば512ビットの2つの素数 p 、 q の積 $N = pq$ と N と互いに素な数 e （ICカードでは、3や、65537が用いられることが多い）をとり、これを公開鍵として公開鍵簿に登録する。このとき、この公開鍵の持ち主Aに送信者Bは、1以上 $N-1$ 以下の数で表現されたデータ（平文） L を、

$$y = L^e \bmod N$$

として暗号化して送信する。ここで、 L^e は L の e 乗を表す記号とする。

この暗号文 R を受け取った持ち主Aは、 $xe \bmod (p-1)(q-1) = 1$ となる秘密鍵 x を用いて

$$y^x \bmod N$$

を計算する。ここで、 $(p-1)(q-1)$ は、 N のオイラー関数の値 $\phi(N)$ である。これは

、 N と互いに素な自然数の個数に等しい。オイラーの定理によれば、

$$y^{((p-1)(q-1))} \bmod N = 1$$

が成り立ち、一方で、 $xe = 1 + k(p-1)(q-1)$ (k は整数) と書くことができるので、

$$\begin{aligned} y^x \bmod N &= L^{(xe)} \bmod N \\ &= L^{(1+k(p-1)(q-1))} \bmod N \\ &= L * L^{(k(p-1)(q-1))} \bmod N \\ &= L \end{aligned}$$

が成り立つ。従って、 $y^x \bmod N$ を計算することで、持ち主 A は、送信者 B の平文 L を復号することができる。この際、秘密鍵 x を計算するのに、 N の素因数 p 、 q が用いられており、現在のところ、素因数分解を介さないで、 x を計算する方法は知られておらず、大きな素数の積を因数分解することは、現実的でない時間が必要であるので、 N を公開しても、 A の秘密鍵は安全である。

【0024】

RSA 暗号の暗号化/復号化操作で用いられるべき乗剰余計算の実装方法の代表的なものは、アディション・チェイン方式とスライディング・ウィンドウ方式である。

まず、アディション・チェイン方式による計算アルゴリズムについて図4を用いて説明する。この方式は、最も多く利用されているものである。ここでは、 $y^x \bmod N$ の計算を、秘密鍵 x のビットを2ビット毎に区切り、上位から読んでいき、それが、00, 01, 10, 11のいずれであるかに応じて、 $y[0] = 1$, $y[1] = y$, $y[2] = y^2 \bmod N$, $y[3] = y^3 \bmod N$ を対応させ、剰余乗算計算を行うことによって実現したものである。勿論、2ビット毎に区切ることは説明の便宜のためであり、実際には1ビット、3ビット、4ビットをまとめて計算することもある。その際も考え方は全く同じである。

【0025】

まず、ビットテーブル $y[0] = 1$, $y[1] = y$, $y[2] = y^2 \bmod N$, $y[3] = y^3 \bmod N$ を用意する(ステップ401)。続いて初期化(ステップ402)を行い、条件分岐(ステップ403)にて、指数 x の最後まで処理し終えたかを判定し、終了

していれば、処理終了し、終了していなければ、4乗の計算（ステップ404）を行う。この4乗計算（ステップ404）は、 x のビットと無関係に行われるが、その次に行われる剰余乗算計算においては、条件分岐処理（ステップ405, 406, 407, 408）が行われ、それぞれの条件に対応して、ステップ409, 410, 411, 412の剰余乗算演算が行われる。

【0026】

この方式で正しい計算が行なえることを簡単に数値例で確認する。この計算方式の本質的部分は指数部分であるので、例として、指数部分のみ数値である例として

$$S = y^{219} \bmod N$$

を取り上げる。ここで、219は二進数表示で11011011である。これをもとに、2ビット幅のアディション・チェーン方式で計算する。11011011を2ビットブロックに分割すると、11 01 10 11である。 $S = 1$ （初期化）を行ない、これを N を法として4乗する。勿論、1を4乗しても1である。次に指数部の先頭のビットブロックを読む。これは、11であるから、 $y[3] = y^3 \bmod N$ を乗じて、 $S = y^3 \bmod N$ となる。次に、再びループし、これを N を法として4乗すると、 $y^{12} \bmod N$ となる。指数部のビットブロックの先頭から2番目のビットブロックを読むと、これは、01であるから、 $y[1] = y$ を乗じて、 $S = y^{13} \bmod N$ となる。再びループし、これを N を法として4乗すると、 $S = y^{52} \bmod N$ となる。指数部のビットブロックの先頭から3番目のビットブロックを読むと、これは、10であるから、 $y[2] = y^2 \bmod N$ を乗じて、 $S = y^{54} \bmod N$ となる。再びループし、これを N を法として4乗すると、 $S = y^{216} \bmod N$ となる。指数部のビットブロックの先頭から4番目のビットブロックを読むと、これは、11であるから、 $y[1] = y^3 \bmod N$ を乗じて、 $S = y^{219} \bmod N$ となる。これが求める答である。

【0027】

次に、もう一つの代表的なべき乗剰余計算アルゴリズムであるスライディング・ウィンドウ方式について図5を用いて説明する。ここでは、 $y^x \bmod N$ を計算する際の x の最大の処理単位を2ビットとする。まず、テーブル $y[2] = y^2 \bmod N$ 、 $y[3] = y^3 \bmod N$ を用意する（ステップ501）。次に S を1に初期化し（ステッ

プ502)、指数xの最後のビットであるかどうかを判定し(ステップ503)、終了していれば、処理を終了し、xの最後のビットでなければ、二乗剰余演算を行う(ステップ504)。指数xの1ビットを読み(ステップ505)、これが1でなければ、再び条件分岐処理(ステップ503)に戻り、この1ビットが1であれば、二乗剰余処理(ステップ507)を行う。次にxの次のビットが受け取れたかどうかを判定し(ステップ508)、受け取れなければ、条件分岐処理(ステップ503)に戻るが、このときはxの最後まで処理が終了しているので、ステップ503の条件はYesとなり、処理を終了する。xの次の処理ビットが受け取れた場合は、条件分岐処理(ステップ509)にて、そのビットが0であるか1であるかに応じ、それぞれ、乗算剰余計算(ステップ510, 511)を実行し、再び条件分岐処理(ステップ503)に戻る。

【 0 0 2 8 】

これも、数値例で簡単に確認しておく。例として

$$S = y^{2226} \bmod N$$

を計算するにあたり、テーブル $y[2] = y^2 \bmod N$, $y[3] = y^3 \bmod N$ を用意する。この際、 $y[0]$, $y[1]$ は不要である。2226は、2進法で、100010110010と書くことができる。これを上位より読んで、2ビットが1を上位に持っているときは、それを一塊とみなし、0が続いているときは、2乗剰余演算を行なうとみなす。つまり、100010110010 = 10 0 0 10 11 0 0 10 という分解に対応するように処理する。従って、 $S=1$ と初期化した後、まず、10に対応する処理、すなわち、 S の2乗剰余演算を行なって、 S に $y[2]$ を掛ける。このとき、 $S = y[2] = y^2 \bmod N$ とする。次に、0に対応した処理、すなわち2乗剰余演算を行なって、 $S = y^4 \bmod N$ を得る。続く0に対応する処理、 $S = y^8 \bmod N$ を行ない、次に、10に対応する処理を行なえば、 $S = ((y^8 \bmod N)^4 \bmod N * y^2 \bmod N) \bmod N = y^{34} \bmod N$ となる。続く処理は、11に対応した、 $S = ((y^{34} \bmod N)^4 * y^3 \bmod N) \bmod N = y^{139} \bmod N$ を実行する。さらに続く2つの0に対応して4乗剰余演算を行ない、 $S = (y^{139} \bmod N)^4 \bmod N = y^{556} \bmod N$ とし、最後の10に対応する処理 $S = ((y^{556} \bmod N)^4 \bmod N * y^2 \bmod N) \bmod N = y^{2226} \bmod N$ によって、求める答を得る。この方式は、テーブルとして、指数ビットの先頭が1であるものだ

けを利用するので、RAMの容量が半分で済むという利点がある。

【0029】

上記のアディション・チェイン方式、スライディング・ウィンドウ方式は、モンゴメリ法と呼ばれる方式を用いて実行されることがあることに注意しておく。モンゴメリ法とは、剰余乗算演算 $AB \bmod N$ を高速に実行する方式であり、特にハードウェア化に向いた方法である。簡単にそのしくみを説明する。詳細は、モンゴメリの原著論文 “Modular Multiplication Without Trial Division”、Mathematics of Computation 44, 170, pp.519-521(1985))に記載されている。

【0030】

モンゴメリ法の本質は、殆ど全てのコンピュータにおいて、 $\bmod 2^n$ の演算は、上位ビットを無視することにより実現できるということを利用することにある。すなわち、 $AB \bmod N$ の計算を2のべきを法とする演算に置き換えることに、その本質がある。RSA暗号では、 N は、大きな素数の積であるから奇数であり、従って任意の2のべきと互いに素である。そこで、 M 、 W を未知数とした不定方程式：

$$AB + MN = WR$$

を考える。ここで、 A 、 B のビット長は n であるものとし、 $R = 2^n$ とする。このとき、この不定方程式は無限個の解を有する。このような M を見つければ、 W は $ABR^{-1} \bmod N$ と合同である。 M は、 R おきに並んでいるので、 R より小さい非負の値がとれる。このとき、 W は、 $ABR^{-1} \bmod N$ であるかまたは、 $ABR^{-1} \bmod N + N$ である。後者の場合は、 N を引き算して求める答を得る。

【0031】

このように、モンゴメリ法においては、 $ABR^{-1} \bmod N$ の形で演算が行なわれるので、上記のアルゴリズム、例えば、アディション・チェイン方式では図4の401、スライディング・ウィンドウ方式では図5の501で実行されるテーブル作成処理において、それぞれのテーブルの値を、 $y[0] = R \bmod N$ 、 $y[1] = y \cdot R \bmod N$ 、 $y[2] = y^2 \cdot R \bmod N$ 、 $y[3] = y^3 \cdot R \bmod N$ に置き換えて処理する。 S の初期値も $R \bmod N$ とする。すると、被乗数 A と乗数 B の値は、もとの値の R 倍になっているため、 $ABR^{-1} \bmod N$ の処理では、必ず R 倍の項が残ることになる。このデー

タ形式をモンゴメリフォーマットと呼ぶことにすれば、アディション・チェイン、スライディング・ウィンドウ両方式を、モンゴメリフォーマットで実行し、最後に、 $R^{(-1)} \bmod N$ 倍することにより、求める答を得ることができる。

【0032】

次に、後の説明の便宜のため、乗算剰余処理におけるオーバーフロー処理について簡単に説明しておく。オーバーフロー処理は、演算装置の実装方法によって若干の違いがあるが、ここでは代表的なものを説明する。最も単純な方式は、 $A*B$ ははじめに求め、これが N 未満であれば、この $A*B$ を答えとし、 N より大きい場合は、 N 未満の値になるまで N を引き算することによって $A*B \bmod N$ を求めるというものである。つまり、この場合、「オーバーフロー」とは、 $A*B$ が N 以上になることであり、「オーバーフロー処理」とは、 N 未満の値になるまで N を引き算するという操作に他ならない。この方式は、 $A*B$ のビット長が小さいときはコード量も少なく、処理時間も無視できるが、ビット長が大きくなると、ICカードのような非力なデバイスでは処理時間が極めて大きくなるため、あまり用いられない。

【0033】

もう一つは、モンゴメリ法で生ずるオーバーフローである。先に簡単に説明したように、モンゴメリ法では、法 N (奇数) における剰余を計算するよりも、法 $R = 2^n$ (n は A, B のビット数) における剰余を求める方が計算が速いことに着目し、不定方程式：

$$AB + MN = WR$$

によって、 $A*B*R^{(-1)} \bmod N$ の計算を上記不定方程式の $M = -A*B*N^{(-1)} \bmod R$ を求める操作に変換する。 M は、 0 以上 R 未満の範囲に一つ存在する。これを求めた場合、 $A*B*R^{(-1)} \bmod N$ が確定するかというと、そうではない。(以下、この不定方程式をモンゴメリ方程式と呼ぶことにする。)

A, B が既に法 N での剰余であれば、 $A < N < R$ かつ $B < N < R$ である。また、 $M < R$ であるので、

$$W = (AB + MN)/R < (NR + RN)/R = 2N$$

となる。つまり、 W は $2N$ よりも小さいが、 N より大きい可能性がある。実際、そのような場合があり、その際には、 N を一回引き算する必要がある。つまり、この

場合、「オーバーフロー」とは、 W が N 以上になることであり、「オーバーフロー処理」とは、 N を一回引く操作に対応する。

【0034】

RSA暗号の場合、乗算剰余処理が出現することは明らかであるが、楕円曲線暗号に対しては、自明な事柄ではないので、以下、楕円曲線暗号について簡単に説明しておく。楕円曲線とは、体 K の上で定義された3次多項式の零点集合であり、 K の標数が2でない場合は、

$$y^2 = x^3 + ax + b$$

という標準形を持つ。標数が2の体の上では、

$$y^2 + cy = x^3 + ax + b \text{ 又は、}$$

$$y^2 + xy = x^3 + ax + b$$

という標準形を持つ曲線である。(いずれの場合も、後に説明する無限遠点 O を含めて考える)。楕円曲線の形状は、図6のようなものになる。

【0035】

暗号で必要なのは、定義体が有限体(ガロア体)の場合のみであるので、その場合に限って説明する。有限個の元からなる体を有限体又はガロア体といい、その構造はよく知られている。その最も単純な構成法は以下の通りである。まず、素数 p を法とする整数環の剰余環 Z_p を考える。 Z_p は集合としては、 $\{0, 1, 2, \dots, p-1\}$ に一致する。 Z_p における和(+)、積(*)は、

$$A (+) B = (A+B) \bmod p$$

$$A (*) B = (A*B) \bmod p$$

で定義されている。 0 以外の元は、(*)に関して逆元を持つので、体の構造を持っている。これを素体といい、 $GF(p)$ と書く。これが最も原始的な有限体の例である。以下、混乱がない限り、(+)は通常の+と書き、(*)も、通常の積 $*$ で表す。積については、 $A*B$ を、 AB のように書くこともある。

【0036】

次に、 $GF(p)$ の元を係数に持つ多項式 $f(X)$ を考え、その零点のうち、 $GF(p)$ に含まれないものを $GF(p)$ に添加することによって、新しい体を構成することができる。これを、 $GF(p)$ の有限次代数拡大体という。 $GF(p)$ の有限次代数拡大体の元

の個数は、 p のべきになっていることが知られている。その元の個数を p^n と書くとき、有限次代数拡大体を $GF(p^n)$ などに表示することがある。

【0037】

準同型定理（松坂和夫「代数系入門」岩波書店、p.125 定理4）によれば、 $GF(p^n)$ は、 $GF(p)$ 係数の多項式全体がつくる環 $GF(p)[X]$ を $GF(p)[X]$ の n 次既約多項式 $f(X)$ の倍元全体の作るイデアルに関する剰余環 $GF(p)[X]/(f(X))$ と同型である。この際、 $f(X)$ は既約でさえあれば、環としては同じものである。従って、適当に n 次の既約多項式を決めて、和と積を、 Z_p と同じように、法 $f(X)$ における和、積とすることで $GF(p^n)$ の演算をマイクロコンピュータ上に実現することができる。

【0038】

ICカードの上で実装する場合は、特に素体 $GF(p)$ (p は素数)の場合とバイナリ体 $GF(2^n)$ の場合が重要である。 $GF(p^n)$ を、 p の大きさをマイクロコンピュータのレジスタの大きさ程度にすることによって、実現する方法もあり、研究が進められている。

【0039】

楕円曲線上の点の間には、演算を定めることができる。図7に示すように、楕円曲線上の二つの点、 P 、 Q があるとき、この二点を通る直線を引き ($P=Q$ のときは接線を引く)、この直線が再び楕円曲線と交わる点 R を x 軸に関して対称に折り返した点は、曲線の対称性から、再び楕円曲線上の点となる。この点を $P+Q$ と書き、 P と Q の「和」と定義する。交わる点がない場合は、架空の点として無限遠点というものを考え、この架空の点で交わっているものとみなす。無限遠点を0と書く。また、楕円曲線上の点 P と x 軸に関して対称な位置にある点を P の逆元といい、 $-P$ で表す。楕円曲線上の一点 P を k 個加えたものを、 kP 、 $-P$ を k 個加えたものを $-kP$ と書いて、 P のスカラー倍という。これらの座標は、 P 、 Q の座標の有理式で表すことができ、従って一般の体の上でこの演算を考えることができる。この「加法」は、通常の加法と同様に、結合法則、交換法則が成立し、この加法に関して、無限遠点0は、通常の数での演算と同様にゼロの役割を果たし、 $-P$ は、 P を加えると、0になる。これは楕円曲線上の加法演算が、可換群（アーベル群）の

構造を持つことを示している。これをモデル・ヴェイユ群ということがある。楕円曲線 E 、定義体 $GF(q)$ ($q=p^n$)を固定したときのモデル・ヴェイユ群を、 $G(E/GF(q))$ と書くことがある。 $G(E/GF(q))$ の構造は非常に単純で、巡回群か、または二つの巡回群の直積と同型になることが知られている。暗号学上は、巡回群になっているか、または2つの巡回群の直積になっている場合は、いずれかの位数(元の数)が、大きな素数で割れることが望ましい。

【0040】

楕円曲線上の点 $P=(x_1, y_1)$ 、 $Q=(x_2, y_2)$ の和 $P+Q$ の座標を (x_3, y_3) とする。 $P+Q=0$ でないとき、以下の関係式が成立する。ICカードでは、標数3を用いる積極的理由は何もないので、以下標数3の場合は省略する。以下、考える楕円曲線は、非特異(non-singular)であるものとする。

【0041】

標数が2, 3でない場合：

$$y^2 = x^3 + ax + b$$

に対しては、

$$x_3 = h^2 - x_1 - x_2$$

$$y_3 = h(x_1 - x_3) - y_1$$

ここで、 h は、

$$h = (y_2 - y_1)/(x_2 - x_1) \text{ if } P \neq Q, \quad (3x_1^2 + a)/(2y_1) \text{ if } P = Q$$

である。

【0042】

標数が2の場合：

$$y^2 + xy = x^3 + ax + b$$

に対しては、

$$x_3 = h^2 + h + x_1 + x_2 + a$$

$$y_3 = h(x_1 + x_3) + x_3 + y_1$$

ここで、 h は、

$$h = (y_1 + y_2)/(x_1 + x_2) \text{ if } P \neq Q, \quad x_1 + (y_1/x_1) \text{ if } P = Q$$

である。

【0043】

上記の和の公式では、演算をガロア体 $GF(p^n)$ の上で行う必要がある。従って、例えば、 $GF(p)$ (p は大きな素数)の上では、

$$x_3 = (h^2 - x_1 - x_2) \bmod p$$

$$y_3 = (h(x_1 - x_3) - y_1) \bmod p$$

を行うことになる。

【0044】

また h の計算も $\bmod p$ で実行しなければならない。つまり、例えば、 $h = (y_2 - y_1)/(x_2 - x_1)$ は、 $h = (y_2 - y_1) * \{(x_2 - x_1)^{(-1) \bmod p}\} \bmod p$ と解釈される。

【0045】

$GF(p^n)$ の場合も、先に述べたように、結局は、多項式の乗算剰余演算を行うことになるので、同様に理解することができる。

【0046】

一般に、 $kP=Q$ の値がわかって、逆に k の値を知るのは計算量が膨大になるため、容易でない。これを楕円曲線上の離散対数問題という。楕円曲線暗号では、楕円曲線上の離散対数問題が困難であることを利用している。楕円曲線を利用した暗号方式には種々のものがあるが、ここでは、特に、楕円ElGamal方式を説明する。楕円曲線 E とその上の一点（一般に大きな位数を持つ点。ベースポイントと呼ぶ） P が公開されているものとする。

【0047】

A氏が、B氏に秘密情報 M （楕円曲線上の点で表現する。平文（暗号文）の楕円曲線上の埋め込みについては、N.コブリッツ、櫻井幸一訳「数論アルゴリズムと楕円暗号入門」シュプリンガーフェアラーク p.253に説明されている）を送信することを考える。

ステップ 1. 受信者B氏は、正整数 $x[B]$ を選び、これを秘密鍵として保持し、

$$Y[B] = x[B]P$$

を公開鍵簿に登録する。

ステップ 2. 送信者A氏は、乱数 r を用いて、

$$C1 = rP$$

$$C2 = M + rY[B]$$

をB氏に送信する。

ステップ 3. 受信者B氏は、C1、C2を受け取り、自分の秘密鍵 $x[B]$ を用いて、

$$C2 - x[B]C1 = M$$

としてMを復元する。

【0048】

楕円ElGamal暗号に限らず、楕円曲線暗号においては、楕円曲線上の点のスカラー倍を計算する必要がある。楕円曲線上の点のスカラー倍を求めるアルゴリズムは、べき乗剰余計算のアルゴリズムに類似している。 kP (k は正整数)を計算する標準的なアルゴリズムをべき乗剰余計算と同様に2ビットずつまとめて処理する場合について示したものを、図8に示す。まず、2ビットの処理を一度に行うために、ベースポイントPのテーブルを作る。べき乗剰余演算においては、mod n での0乗、1乗、2乗、3乗に対応して、0(無限遠点)、 P 、 $2P$ 、 $3P$ を用意する(ステップ801)。べき乗剰余演算とは異なり、このテーブルは毎回書き換える必要はなく、あらかじめ用意しておくことができる。次に、計算用の点の値を初期化する(ステップ802)。次に点Sの2倍を計算し(ステップ803)した後、 k の最後のビットまで実行したかどうか判定し(ステップ804)、実行済でなければ、 k のビットの値(2ビットごと)によって条件分岐して(ステップ804、805、806、807)、これらに対応する点 $P[0]$ 、 $P[1]$ 、 $P[2]$ 、 $P[3]$ を加える(ステップ808、809、810、811)。この処理を k のビットが尽きるまで続けることにより、 kP を計算することができる。この計算は k の上位から2ビットずつ区切って見て計算する方式である。べき乗剰余計算と、数学的には同一の構造をしていることがわかる。スライディング・ウィンドウ方式にすることも容易である。後に再び説明するが、RSAにおけるべき乗剰余演算、楕円曲線上の加法演算は、それぞれ、 Z_n 、 $G(E/GF(q))$ という代数系の上で行われる演算と考えることができる。

【0049】

一方で、マイクロコンピュータが内部のプログラムを実行する際、動作時の内部消費電力が漏洩する可能性があるため、このプロセスをマイクロコンピュータ

で実現する場合、秘密鍵の処理が漏洩し、危険に曝されることになる。例えば、 k のビット（この例では2ビット毎）の違いにより、分岐が行われるので、もしも、その処理が消費電流の差として現れれば、電流波形から k のビットを特定することができる可能性がある。

【 0 0 5 0 】

上記の背景を踏まえて、本発明の実施例を説明する。図9に示した処理は、本発明の典型的応用例である。本実施例は、べき乗剰余計算 $y^x \bmod N$ をアディクション・チェイン方式によって実行するものである。但し、本実施例では、モンゴメリ法による乗算剰余処理を用いて、RSA暗号のべき乗剰余計算を行う。この処理において、 $A \cdot B \cdot R^{-1} \bmod N$ という乗算剰余演算が用いられる。先に述べたのと同じく、 N は奇数（一般にRSA暗号では、 N は、大きな素数 p 、 q の積であり、奇数である）であり、 $R = 2^n$ (n はデータのビット長)とする。

【 0 0 5 1 】

まず、ステップ901にて、 $S = R \bmod N$ とし、さらに、 x のビットブロックの番号を数えるためのカウンタ j を0に初期化し、条件分岐判定用の変数 $v[0] = 0$ とする。次に2ビットアディクション・チェイン方式処理用のテーブル作成を行う（ステップ902）が、この際、通常のアディクション・チェイン処理に必要なオリジナルテーブル(903) $y[0][0] = R \bmod N$, $y[1][0] = yR \bmod N$, $y[2][0] = y^2R \bmod N$, $y[3][0] = y^3R \bmod N$ の他に、モジュラス N における反転テーブル(904) $y[0][1] = N - (R \bmod N)$, $y[1][1] = N - (yR \bmod N)$, $y[2][1] = N - (y^2R \bmod N)$, $y[3][1] = N - (y^3R \bmod N)$ をも用意する。テーブルは通常、RAM上に存在する。このテーブル作成の後、条件分岐処理（ステップ905）にて、指数 x が最後まで読まれたか（全てのビットブロックを読んだか）を調べる。指数 x の全てのビットブロックの処理が終了していなければ、ステップ906に進み乱数 $v[j]$ を発生する（ステップ906）。この乱数は、0または1である。次に、 $S = S^{2^{v[j]}} \cdot R^{-1} \bmod N$ を2回実行する（ステップ907）。最初の処理では、 $S = R \bmod N$ に初期化されているので、この二乗剰余演算の結果は、 $S = (R \bmod N)^{2^{v[j]}} \cdot R^{-1} \bmod N = R \bmod N$ となり、ステップ907の処理結果は、 $R \bmod N$ となることに注意する。次に、条件分岐処理（ステップ908, 909, 910, 911）において、指数 x

の2ビットブロックを読み、このビットブロック（図中では「xのビットブロック」と表現されている）が、2進数で、00であるか01であるか、10であるか、11であるかに応じて、それぞれ、ステップ912, 913, 914, 915に進み、乗算剰余演算を行う。この乗算剰余演算では、 $v[j]$ が、0であるか、1であるかに応じて、用いるテーブル値が異なる。例えば、xのビットブロックが10であれば、通常は、 $S = S * y[2][0] * R^{(-1)} \bmod N = S * (y^{2R} \bmod N) \bmod N$ を計算するが、 $v[j] = 1$ のときは、反転テーブルから、 $y[2][1]$ を取り出して、 $S = S * y[2][1] * R^{(-1)} \bmod N = S * (N - (y^{2R} \bmod N)) * R^{(-1)} \bmod N$ を計算する（ステップ914）。ステップ912, 913, 915の処理においても、同様に $v[j]$ の値に応じて乗ずる値を変える。乗算剰余処理を終えた後、ステップ921にて、カウンタをインクリメントし、ステップ905の処理に戻る。条件分岐処理（ステップ905）にて、指数xのビットブロックを全て読み終わっている場合には、ステップ916の処理に進む。ステップ916では、モンゴメリ形式（ $R \bmod N$ が掛かっている）のデータを通常の値に戻すために、 $R^{(-1)} \bmod N$ を乗ずる処理を行う。次に、 $T = N - S$ を計算し、これをRAMに置く（ステップ917）。ここで、SとTはRAM上で異なる位置に配置され、RAM上で重複することはないものとする。条件分岐処理（ステップ918）にて、最後のvの値が1であれば、Tを出力し（ステップ919）、 $v = 0$ であれば、Sを出力する（ステップ920）。この処理によって正しい値が出力されることは、 $(tN - B)^2 \bmod N = B^2 \bmod N$ （tは整数）であることから明らかである。このように処理することによって、オーバーフローの処理が本来のものとは異なるものになるため、ICチップの消費電力や、処理時間等を観測して内部処理を行うことが困難になる。これは、乗算剰余演算 $A * B \bmod N$ のAを $sN + A * (-1)^f$ 、Bを $tN + B * (-1)^g$ （s, t, f, gは整数）に置き換えたときに、特に（s, t, f, g）= {(0, 0, 0, 0), (0, 1, 0, 1)}としたものの一つである（{ }内は集合の要素）。

【0052】

上記実施例では、各jに関して $v[j]$ を乱数として変化させているが、全ての $j = 0, 1, \dots, m-1$ （mはビットブロックの数）を0とするか、1とするかを、最初にランダムに決定するという方法もある。図10に、これを示す。まずステップ10

01にて、 $S = R \bmod N$ とし、さらに、 x のビットブロックの番号を数えるためのカウンタ j を0に初期化する。次に2ビットアディション・チェイン方式処理用のテーブル作成を行う（ステップ1002）が、この際、通常のアディション・チェイン処理に必要なオリジナルテーブル(1003) $y[0][0] = R \bmod N$, $y[1][0] = yR \bmod N$, $y[2][0] = y^2R \bmod N$, $y[3][0] = y^3R \bmod N$ の他に、モジュラス N における反転テーブル(1004) $y[0][1] = N - (R \bmod N)$, $y[1][1] = N - (yR \bmod N)$, $y[2][1] = N - (y^2R \bmod N)$, $y[3][1] = N - (y^3R \bmod N)$ をも用意する。テーブルは通常、RAM上に存在する。このテーブル作成の後、ステップ1005に進み乱数 v を発生する。この乱数は、0または1である。次に条件分岐処理（ステップ1006）にて、指数 x が最後まで読まれたか（全てのビットブロックを読んだか）を調べる。次に、 $S = S^2 \cdot R^{(-1)} \bmod N$ を2回実行する（ステップ1007）。最初の処理では、 $S = R \bmod N$ に初期化されているので、この二乗剰余演算では、 $S = (R \bmod N)^2 \cdot R^{(-1)} \bmod N = R \bmod N$ となり、ステップ1007の処理結果は、 $R \bmod N$ となることに注意する。次に、条件分岐処理（ステップ1008, 1009, 1010, 1011）において、指数 x の2ビットブロックを読み、このビットブロックが、2進数で、00であるか01であるか、10であるか、11であるかに応じて、それぞれステップ1012, 1013, 1014, 1015に進み、乗算剰余演算を行う。乗算剰余演算では、 v が、0であるか、1であるかに応じて、用いるテーブル値が異なる。例えば、 x のビットブロックが10であれば、通常は、 $S = S \cdot y[2][0] \cdot R^{(-1)} \bmod N = S \cdot (y^2R \bmod N) \bmod N$ を計算するが、 $v = 1$ のときは、反転テーブルから $y[2][1]$ を取り出して、 $S = S \cdot y[2][1] \cdot R^{(-1)} \bmod N = S \cdot (N - (y^2R \bmod N)) \cdot R^{(-1)} \bmod N$ を計算する（ステップ1014）。ステップ1012, 1013, 1015の処理においても、同様に v の値に応じて乗ずる値を変える。乗算剰余処理を終えた後、ステップ1021にて、カウンタをインクリメントし、ステップ1006の処理に戻る。条件分岐処理（ステップ1006）にて、指数 x のビットブロックを全て読み終わっている場合には、ステップ1016の処理に進む。ステップ1016では、モンゴメリ形式（ $R \bmod N$ が掛かっている）のデータを通常の値に戻すために、 $R^{(-1)} \bmod N$ を乗ずる処理を行う。次に、 $T = N - S$ を計算し、これをRAMに置く（ステップ1017）。ここで、 S と T はRAM上で異なる位置に配置され、RAM上で重複す

ることではないものとする。条件分岐処理（ステップ1018）にて、最後の v の値が1であれば、 T を出力し（ステップ1019）、 $v = 0$ であれば、 S を出力する（ステップ1020）。この処理によって正しい値が出力されることは、 $(tN - B)^2 \bmod N = B^2 \bmod N$ （ t は整数）であることから明らかである。このように処理することによって、オーバーフローの処理が本来のものとは異なるものになるため、ICチップの消費電力や、処理時間等を観測して内部処理を行うことが困難になる。これは、乗算剰余演算 $A \cdot B \bmod N$ の A , B を上記のように置き換えたときに、特に $(s, t, f, g) = \{(0, 0, 0, 0), (0, 1, 0, 1)\}$ としたものの一つである。

【0053】

上記の2つの実施例では、モンゴメリ法を用いたものを例として示したが、モンゴメリ法を用いない形式で本発明を適用することも容易である。この例を示そう。図9の実施例を非モンゴメリ形式に変更することは全く容易であって、モンゴメリ・フォーマットに変換している部分を除去すればよい。図11は、モンゴメリ法を用いない通常の実装方法を示したものである。

【0054】

まずステップ1101にて、 $S = 1$ とし、さらに、 x のビットブロックの番号を数えるためのカウンタ j を0に初期化し、条件分岐判定用の変数 $v[0] = 0$ とする。次に2ビットアディション・チェイン方式処理用のテーブル作成を行う（ステップ1102）が、この際、通常のアディション・チェイン処理に必要なオリジナルテーブル(1103) $y[0][0] = 1$, $y[1][0] = y \bmod N$, $y[2][0] = y^2 \bmod N$, $y[3][0] = y^3 \bmod N$ の他に、モジュラス N における反転テーブル(1104) $y[0][1] = N - 1$, $y[1][1] = N - (y \bmod N)$, $y[2][1] = N - (y^2 \bmod N)$, $y[3][1] = N - (y^3 \bmod N)$ をも用意する。テーブルは通常、RAM上に存在する。このテーブル作成の後、条件分岐処理（ステップ1105）にて、指数 x が最後まで読まれたか（全てのビットブロックを読んだか）を調べる。指数 x の全てのビットブロックの処理が終了していなければ、ステップ1106に進み乱数 $v[j]$ を発生する。この乱数は、0または1である。次に、 $S = S^2 \bmod N$ を2回実行する（ステップ1107）。次に、条件分岐処理（ステップ1108, 1109, 1110, 1111）において、指数 x の2ビ

ットブロックを読み、このビットブロック（図中では「xのビットブロック」と表現されている）が、2進数で、00であるか01であるか、10であるか、11であるかに応じて、それぞれ、ステップ1112, 1113, 1114, 1115に進み、乗算剰余演算を行う。乗算剰余演算では、 $v[j]$ が、0であるか、1であるかに応じて、用いるテーブル値が異なる。例えば、xのビットブロックが10であれば、通常は、 $S = S * y[2][0] \bmod N = S * (y^2 \bmod N) \bmod N$ を計算するが、 $v[j] = 1$ のときは、反転テーブルから、 $y[2][1]$ を取り出して、 $S = S * y[2][1] \bmod N = S * (N - (y^2 \bmod N)) \bmod N$ を計算する（ステップ1114）。ステップ1112, 1113, 1115の処理においても、同様に $v[j]$ の値に応じて乗ずる値を変える。乗算剰余処理を終えた後、ステップ1120にて、カウンタをインクリメントし、ステップ1105の処理に戻る。条件分岐処理（ステップ1105）にて、指数xのビットブロックを全て読み終わっている場合には、ステップ1116の処理に進む。ステップ1116では、 $T = N - S$ を計算し、これをRAMに置く。ここで、SとTはRAM上で異なる位置に配置され、RAM上で重複することはないものとする。条件分岐処理（ステップ1117）にて、最後のvの値が1であれば、Tを出力し（ステップ1118）、 $v = 0$ であれば、Sを出力する（ステップ1119）。この処理によって正しい値が出力されることは、 $(tN - B)^2 \bmod N = B^2 \bmod N$ （tは整数）であることから明らかである。このように処理することによって、オーバーフローの処理が本来のものとは異なるものになるため、ICチップの消費電力や、処理時間等を観測して内部処理を行うことが困難になる。これは、乗算剰余演算 $A * B \bmod N$ のA, Bを上記のように置き換えたときに、特に $(s, t, f, g) = \{(0, 0, 0, 0), (0, 1, 0, 1)\}$ としたものの一つである。

【0055】

以下図11の実施例に対応する発明を方法ステップの形で要約しておく。

【0056】

情報処理装置を利用して、暗号処理中に出現する剰余乗算である $A * B \bmod N$ を計算する方法であって、その方法は、

- (1) $S_1 = A * B \bmod N$ を計算するステップ、
- (2) (1)の代わりに $S_2 = \{sN + A * (-1)^f\} * \{tN + B * (-1)^g\}$

$\text{mod } N$ (s, t, f, g は少なくとも1つは0でない整数であり、 f, g は0又は1)を計算するステップ、

(3) 上記(1)と(2)のいずれかを適宜選択するステップ、

(4) 上記(1)(2)(3)を適宜繰り返し、最後に上記(1)を選択したときはその計算結果 S_1 を出力し、上記(2)を選択したときには S_2 の代わりに $N - S_2$ を出力するステップ、および

(5) S_1 と $N - S_2$ を、暗号処理の剰余乗算 $A * B \text{ mod } N$ の計算結果として用いるステップを有することを特徴とする耐タンパーモジュラ演算処理方法。

【0057】

上記の実施例(図9、図10及び図11の実施例)においては、 v として乱数を用いているが、これを0と1を交互に取るような仕方で変化させたり、疑似乱数や、カオス的数列といったものに変えることができることは言うまでもない。また、上記実施例では、 s, t, f, g を二通りにしか変化させていないが、より多くの変数を変化させることにより、攪乱効果を上げることができる。この場合の実施例を図12に示す。

【0058】

まずステップ1201にて、 $S = 1$ とし、さらに、 x のビットブロックの番号を数えるためのカウンタ j を0に初期化し、条件分岐判定用の変数 $v[0] = 0$ とする。次に、4つの乱数 $w[k]$ ($k=0, 1, 2, 3$)を用意する(ステップ1202)。この乱数は、 $0 < w[k] < \text{MAX}$ を満たす整数で、 MAX は、メモリやレジスタの大きさ等の制約によって決める。次に2ビットアディション・チェイン方式処理用のテーブル作成を行う(ステップ1203)が、この際、通常のアディション・チェイン処理に必要なオリジナルテーブル(1204) $y[0][0] = 1, y[1][0] = y \text{ mod } N, y[2][0] = y^2 \text{ mod } N, y[3][0] = y^3 \text{ mod } N$ の他に、モジュラス N における反転テーブル(1205) $y[0][1] = N - 1, y[1][1] = N - (y \text{ mod } N), y[2][1] = N - (y^2 \text{ mod } N), y[3][1] = N - (y^3 \text{ mod } N)$ をも用意する。テーブルは通常、RAM上に存在する。このテーブル作成の後、条件分岐処理(ステップ1206)にて、指数 x が最後まで読まれたか(全てのビットブロックを読んだか)を調べる。指数 x の全てのビットブロックの処理が終了していなければ、ステップ1207に進み乱数 $v[j]$ を発生

する。この乱数は、0または1である。次に、 $S = S^2 \bmod N$ を2回実行する（ステップ1208）。次に、条件分岐処理（ステップ1209, 1210, 1211, 1212）において、指数 x の2ビットブロックを読み、このビットブロック（図中では「 x のビットブロック」と表現されている）が、2進数で、00であるか01であるか、10であるか、11であるかに応じて、それぞれ、ステップ1213, 1214, 1215, 1216に進み、乗算剰余演算を行う。乗算剰余演算では、 $v[j]$ が、0であるか、1であるかに応じて、用いるテーブル値が異なる。例えば、 x のビットブロックが10であれば、通常は、 $S = S * y[2][0] \bmod N = S * (y^2 \bmod N) \bmod N$ を計算するが、 $v[j] = 1$ のときは、反転テーブルから、 $y[2][1]$ を取り出して、 $S = S * y[2][1] \bmod N = S * (N - (y^2 \bmod N)) \bmod N$ を計算する（ステップ1215）。ステップ1213, 1214, 1216の処理においても、同様に $v[j]$ の値に応じて乗ずる値を変える。乗算剰余処理を終えた後、ステップ1221にて、カウンタをインクリメントし、ステップ1206の処理に戻る。条件分岐処理（ステップ1206）にて、指数 x のビットブロックを全て読み終わっている場合には、ステップ1217の処理に進む。ステップ1217では、 $T = N - S$ を計算し、これをRAMに置く。ここで、 S と T はRAMで異なる位置に配置され、RAM上で重複することはないものとする。条件分岐処理（ステップ1218）にて、最後の v の値が1であれば、 T を出力し（ステップ1219）、 $v = 0$ であれば、 S を出力する（ステップ1220）。この処理によって正しい値が出力されることは、 $(tN - B)^2 \bmod N = B^2 \bmod N$ （ t は整数）であることから明らかである。このように処理することによって、オーバーフローの処理が本来のものとは異なるものになるため、ICチップの消費電力や、処理時間等を観測して内部処理を行うことが困難になる。これは、本発明の実施例の一つである。これをモンゴメリ方式の場合に変更することは容易であるので、例を挙げることは省略する。

【0059】

これまで示した実施例は、乗数側を変形するものである。本発明の趣旨は、乗数側、被乗数側のいずれか、または両方に適用することによって損なわれることはない。これは数学的には自明な事実である。しかし、実装上は若干の違いが生ずる。乗数側の変形は、最初の段階でテーブルを作成しさえすれば、その後は「

取り出す値」を変化させることによって本発明を実現することができる。ところが被乗数を変形する場合は、被乗数自身が各計算過程で異なるため、計算過程毎に途中計算値 S と $N-S$ を構成しなければならない。これを示すために、被乗数側を変形する実施例を示す。

【 0 0 6 0 】

図 1 3 を順に説明する。まず、ステップ1301にて、 $S = R \bmod N$ とし、さらに、 x のビットブロックの番号を数えるためのカウンタ j を0に初期化し、条件分岐判定用の変数 $v[0] = 0$ とする。次に2ビットアディション・チェイン方式処理用のテーブル作成を行う（ステップ1302）。テーブルは、通常のテーブル値 $y[0] = R \bmod N$, $y[1] = yR \bmod N$, $y[2] = y^2R \bmod N$, $y[3] = y^3R \bmod N$ で構成され、乗数側の変形に用いる反転テーブルは不要である。テーブルは通常、RAM上に存在する。このテーブル作成の後、条件分岐処理（ステップ1303）にて、指数 x が最後まで読まれたか（全てのビットブロックを読んだか）を調べる。指数 x の全てのビットブロックの処理が終了していなければ、ステップ1304に進み乱数 $v[j]$ を発生する。この乱数は、0または1である。次に、 $S = S^2 \cdot R^{(-1)} \bmod N$ を2回実行する（ステップ1305）。最初の処理では、 $S = R \bmod N$ に初期化されているので、この二乗剰余演算では、 $S = (R \bmod N)^2 \cdot R^{(-1)} \bmod N = R \bmod N$ となり、ステップ1305の処理結果は、 $R \bmod N$ となることに注意する。次に、この S から、 $S[0] = S$, $S[1] = N - S$ を計算しRAM上に格納する（ステップ1306）。続いて、条件分岐処理（ステップ1307, 1308, 1309, 1310）において、指数 x の2ビットブロックを読み、このビットブロック（図中では「 x のビットブロック」と表現されている）が、2進数で、00であるか01であるか、10であるか、11であるかに応じて、それぞれ、ステップ912, 913, 914, 915に進み、乗算剰余演算を行う。乗算剰余演算では、 $v[j]$ が、0であるか、1であるかに応じて、用いるテーブル値が異なる。例えば、 x のビットブロックが10であれば、通常は、 $S = S[0] * y[0] * R^{(-1)} \bmod N = S * (y^2R \bmod N) \bmod N$ を計算するが、 $v[j] = 1$ のときは、 $S[0]$ の代わりに $S[1]$ を取り出して、 $S = S[1] * y[2] * R^{(-1)} \bmod N = (N - S) * (y^2R \bmod N) * R^{(-1)} \bmod N$ を計算する（ステップ1313）。ステップ1311、1312、1314の処理においても、同様に $v[j]$ の値に応じて乗ずる値を変える。乗算

剰余処理を終えた後、ステップ1315にて、カウンタをインクリメントし、ステップ1303の処理に戻る。条件分岐処理（ステップ1303）にて、指数 x のビットブロックを全て読み終わっている場合には、ステップ1316の処理に進む。ステップ1316では、モンゴメリ形式（ $R \bmod N$ が掛かっている）のデータを通常の値に戻すために、 $R^*(-1) \bmod N$ を乗ずる処理を行う。次に、 $T = N - S$ を計算し、これをRAMに置く（ステップ1317）。ここで、 S と T はRAM上で異なる位置に配置され、RAM上で重複することはないものとする。条件分岐処理（ステップ1318）にて、最後の v の値が1であれば、 T を出力し（ステップ1319）、 $v = 0$ であれば、 S を出力する（ステップ1320）。この処理によって正しい値が出力されることは、 $(tN - B)^2 \bmod N = B^2 \bmod N$ （ t は整数）であることから明らかである。このように処理することによって、オーバーフローの処理が本来のものとは異なるものになるため、ICチップの消費電力や、処理時間等を観測して内部処理を行うことが困難になる。これは、乗算剰余演算 $A*B \bmod N$ の A , B を上記のように置き換えたときに、特に $(s, t, f, g) = \{(0, 0, 0, 0), (1, 0, 1, 0)\}$ としたものの一つである。これをモンゴメリ法を用いない方式に変更することは容易であるので、例を挙げることは省略する。

【0061】

これまでに示した実施例は、乗数または被乗数を変形したものであるが、両方を変形することもできることは言うまでもない。また、例えば図13のステップ1305の処理過程で、 $S = (N - S)^2 * R^*(-1) \bmod N$ のように処理を行うことにより、消費電流を本来のそれとは異なるものにすることが可能となる。本発明の実施例は、様々なバリエーションを持っているが、本質的には、上記の実施例を組み合わせる構成することが可能である。

【0062】

上記の実施例はいずれも、べき乗剰余計算の処理に対して適用されたものである。べき乗剰余計算に対して本発明はとりわけ有効であるが、上記の考え方をより一般のモジュラ計算（剰余計算）に置き換えることによって、オーバーフロー等の処理から秘密情報が漏洩することを防ぐことも可能である。べき乗剰余計算において本発明が有効である理由は、本質的には、

$$(tN - B)^2 \bmod N = B^2 \bmod N \quad (t \text{ は整数})$$

という関係式が成り立つという事実に着着する。

【0063】

ところがべき乗剰余計算とは異なる処理、例えば、楕円曲線暗号における曲線上のベースポイントPのスカラ倍を計算するというような場合には、上記の関係式のような「自然な」修正処理が現れるとは限らない。この事情を簡単に説明する。素体GF(p) (pは大きな素数)において、楕円曲線E:

$$y^2 = x^3 + ax + b$$

を考える。先に述べたように、E上の点Pのスカラ倍kPを計算する場合は、点の2倍算と、点同士の和の計算が出現する(図8参照)。ここで、この二つの計算をより詳細に分解すると、通常のICカードマイコンにおいては、RSAよりも複雑な計算過程を経なければならないことがわかる。

【0064】

先に述べたように、楕円曲線上の点 $P=(x_1, y_1)$, $Q=(x_2, y_2)$ の和 $P+Q$ の座標を (x_3, y_3) とすると、 $P+Q=0$ でないとき

$$x_3 = (h^2 - x_1 - x_2) \bmod p$$

$$y_3 = (h*(x_1 - x_3) - y_1) \bmod p$$

と表現できる。ここで、hは、

$$h = (y_2 - y_1) * ((x_2 - x_1)^{-1}) \bmod p \text{ if } P \neq Q, \quad (3x_1^2 + a) * ((2y_1)^{-1}) \bmod p \text{ if } P = Q$$

である。

【0065】

この群演算(モデル・ヴェイユ群の群演算)に必要な乗算剰余演算を列挙すると、

$$h = (y_2 - y_1) * (x_2 - x_1)^{-1} \bmod p \text{ if } P \neq Q, \text{ --- (E1)}$$

$$h = (3x_1^2 + a) * (2y_1)^{-1} \bmod p \text{ if } P = Q \text{ --- (E1)'}$$

$$h^2 \bmod p \text{ --- (E2)}$$

$$h * (x_1 - x_3) \bmod p \text{ --- (E3)}$$

となる。(E1)の処理において、 $A = y_2 - y_1$, $B = (x_2 - x_1)^{-1} \bmod p$ (また

は $x_2 - x_1$ をそれぞれ、 $sN + A*(-1)^f$, $tN + B*(-1)^g$ (s , t , f , g は整数)。但し、 f , g は 0 又は 1) に置き換えると、オーバーフローのパターンが変化する ((E1)' の場合も同様)。 h は (E2) の処理において法 p において 2 乗され、この値は本来のものと同一になるが、(E3) の処理では、値が、法 p で反転してしまうため、正しい y_3 を求めるには、(E3) の結果を補正しなければならない。明らかに $f+g$ が偶数であれば補正は不要で、 $f+g$ が奇数の場合は、補正が必要となる。

【 0 0 6 6 】

ここで、実施例を示すが、楕円曲線暗号の大域的処理フローについては、図 8 及び対応する説明において説明したので、ここでは、2 倍計算 $S = 2S$ と $S = S + P[j]$ の計算部に絞って説明を行う。すなわち、図 8 における、ステップ 803 (2 倍計算)、ステップ 809, 810, 811, 812 (和の計算) の処理に本発明の処理方式を実装する方法を説明する。

【 0 0 6 7 】

まず、素体 $GF(p)$ の上での実施例のうち、最も簡単なものを説明する。素体上での計算は、全て、法 p のもとで実行すればよい。図 15 は $GF(p)$ 上での楕円曲線 $E: y^2 = x^3 + ax + b$ における点 $P(x, y)$ の 2 倍計算に本発明を適用したものである。以下、 $2P = (x_3, y_3)$ とする。

【 0 0 6 8 】

ステップ 1401 において、 $D1 = (2*y)^{(-1)} \bmod p$ を計算する。次に、ステップ 1402 にて、 $D2 = (3*x^2 + a) \bmod p$ を計算する。これら $D1$, $D2$ に対して、テーブル $D1[0] = D1$, $D2[0] = D2$, $D1[1] = p - D1$, $D2[1] = p - D2$ を計算して RAM に格納する (ステップ 1403)。 $D1[1]$, $D2[1]$ は、それぞれ、法 p に対する $D1$, $D2$ の反転値である。次に 2 つ (2 ビット) の乱数 v , w (v , w は、共に 0 又は 1) を発生する (ステップ 1404)。次に、ステップ 1403 で求めたテーブルから、 $D1[v]$, $D2[v]$ を取り出して、乗算剰余計算: $h = D1[v] * D2[v] \bmod p$ を実行して結果を RAM に格納する (ステップ 1405)。ここで、 $D1[1] * D2[1] \bmod p = (p - D1) * (p - D2) \bmod p = D1 * D2 \bmod p$ であるので、ステップ 1405 の終了時には、 $v = 0, 1$ いずれの場合でも、正しい h が求まる。次にステップ 1406, 1407 にて、 x_3 を求め、答えを RAM に格納する。ステップ 1408 にて、 $D3 = (x - x_3) \bmod p$ を計算す

る。次に、このD3と、先に用いたhに対して、テーブル $h[0] = h$, $D3[0] = D3$, $h[1] = p - h$, $D3[1] = p - D3$ を求めてRAMに格納する（ステップ1409）。 $h[1]$, $D3[1]$ は、それぞれ、法pに対するh, D3の反転値である。次に、ステップ1409で求めたテーブルから、 $h[v]$, $D3[v]$ を取り出して、乗算剰余計算： $h[w] * D3[w] \bmod p$ を実行して結果をRAMに格納する（ステップ1410）。ここで、 $h[1] * D3[1] \bmod p = (p - h) * (p - D3) \bmod p = h * D3 \bmod p$ であるので、ステップ1410の終了時には、 $w = 0$, 1いずれの場合でも、正しい値が求まる。最後に、ステップ1411にて、 $y3 = (y3 - y) \bmod p$ を求めてRAMに格納する。これで、2Pの各座標の値が求められたことになる。これは、本発明の実施例の一つである。

【 0 0 6 9 】

本実施例にて、ステップ1401における逆数計算を実行する代表的な方法は、拡張ユークリッド互除法を利用するものとフェルマーの小定理を利用するものである。拡張ユークリッド互除法では、不定方程式： $2*y*D1 + p*U = 1$ を($2*y$ とpの最大公約数を求める操作（ユークリッド互除法）を繰り返すことで、結果的にD1を求める方法である。一方、フェルマーの小定理を用いる方法とは、フェルマーの小定理、すなわち、pと互いに素な正の整数gに対し、 $g^{p-1} \bmod p = 1$ が成り立つことから、 $g^{p-2} \bmod p = g^{-1} \bmod p$ となることを用いるものである。つまり、 $D1 = (2*y)^{p-2} \bmod p$ として、D1を求めるものである。フェルマーの小定理を用いる方法では、逆数の計算は、べき乗剰余計算に帰着する。このべき乗剰余計算に本発明の方法を適用することは容易なことであり、オーバーフロー処理からのリーク情報を減らすことができる。

【 0 0 7 0 】

次にGF(p)上の楕円曲線に対する点の和の計算に本発明を適用した場合の実施例を示す。以下、楕円曲線上の点 $P=(x1, y1)$ 、 $Q=(x2, y2)$ の和 $P+Q$ の座標を($x3, y3$)とし、 $P+Q=0$ でないものとする。図16はGF(p)上での楕円曲線 $E: y^2 = x^3 + ax + b$ における点PとQの和の計算に本発明を適用したものである。

【 0 0 7 1 】

条件分岐処理（ステップ1501）にて、PとQが一致しているかどうかを判定する。もし、PとQが一致していれば、2倍算になり、図15について説明した処理と

同じものになるので、説明は省略する（ただし x , y の符号は x_1 , y_1 の符号に読み替える）。以下、 P と Q が一致していない場合の処理を示す。ステップ1502において、 $D1 = (x_2 - x_1)^{-1} \bmod p$ を計算する。次に、ステップ1503にて、 $D2 = (y_2 - y_1) \bmod p$ を計算する。これら $D1$, $D2$ に対して、テーブル $D1[0] = D1$, $D2[0] = D2$, $D1[1] = p - D1$, $D2[1] = p - D2$ を計算してRAMに格納する（ステップ1504）。 $D1[1]$, $D2[1]$ は、それぞれ、法 p に対する $D1$, $D2$ の反転値である。次に二つ（2ビット）の乱数 v , w (v , w は、共に0又は1)を発生する（ステップ1505）。次に、ステップ1504で求めたテーブルから、 $D1[v]$, $D2[v]$ を取り出して、乗算剰余計算： $h = D1[v] * D2[v] \bmod p$ を実行して結果をRAMに格納する（ステップ1506）。ここで、 $D1[1] * D2[1] \bmod p = (p - D1) * (p - D2) \bmod p = D1 * D2 \bmod p$ であるので、ステップ1506の終了時には、 $v = 0$, 1 いずれの場合でも、正しい h が求まる。次にステップ1507, 1509にて、 x_3 を求め、答をRAMに格納する。ステップ1509にて、 $D3 = (x_1 - x_3) \bmod p$ を計算する。次に、この $D3$ と、先に用いた h に対して、テーブル $h[0] = h$, $D3[0] = D3$, $h[1] = p - h$, $D3[1] = p - D3$ を求めてRAMに格納する（ステップ1510）。 $h[1]$, $D3[1]$ は、それぞれ、法 p に対する h , $D3$ の反転値である。次に、ステップ1510で求めたテーブルから、 $h[w]$, $D3[w]$ を取り出して、乗算剰余計算： $h[w] * D3[w] \bmod p$ を実行して結果をRAMに格納する（ステップ1511）。ここで、 $h[1] * D3[1] \bmod p = (p - h) * (p - D3) \bmod p = h * D3 \bmod p$ であるので、ステップ1511の終了時には、 $w = 0$, 1 いずれの場合でも、正しい値が求まる。最後に、ステップ1512にて、 $y_3 = (y_3 - y_1) \bmod p$ を求めてRAMに格納する。これで、 $P+Q$ の各座標の値が求められたことになる。これは、本発明の実施例の一つである。

【0072】

図16の実施例は、容易にガロア体 $GF(p^n)$ (p は素数、 n は正の整数)に拡張することができる。先に述べたように、 $GF(p^n)$ は、 $GF(p)$ 係数の多項式全体がつくる環（多項式環） $GF(p)[X]$ の $GF(p)$ における既約多項式 $\Phi(X)$ (reduction polynomialと呼ばれることもある)の倍元全体が生成する素イデアル(prime ideal) ($\Phi(X)$) による剰余環(quotient ring) $GF(p)[x]/(\Phi(X))$ と同型(homomorphic)であり、計算機では、 $GF(p)[x]/(\Phi(X))$ として実現される。その演算を実行するに

あたり、 $A(X)$, $B(X)$ を $GF(p)[X]$ の元（すなわち、 $GF(p)$ 係数の多項式）とすると、 $GF(p)[X]/(\Phi(X))$ での演算は、

$$(\text{和}) \quad \{A(X) + B(X)\} \bmod \Phi(X)$$

$$(\text{積}) \quad A(X) * B(X) \bmod \Phi(X)$$

とすることによって実現することができる。但し、係数の演算は、 $\bmod p$ で行う。

【0073】

実例を示す。素数 $p = 5$, $n = 2$ とし、reduction polynomial $\Phi(X)$ を $X^2 + X + 1$ とする。 $\Phi(X)$ が既約であることは、 $\Phi(0) = 1$, $\Phi(1) = 3$, $\Phi(2) = 7 \equiv 2(\bmod 5)$, $\Phi(3) = 13 \equiv 3(\bmod 5)$, $\Phi(4) = 21 \equiv 1(\bmod 5)$ であることからすぐに分かる。 $A(X) = 4X^2 + 3X + 2$, $B(X) = 3X^2 + 4X + 1$ とすると、この、二つの多項式の $GF(5)[X]/(X^2+X+1)$ における和と積は、

$$(\text{和}) \quad \{A(X) + B(X)\} \bmod \Phi(X)$$

$$= 7X^2 + 7X + 3$$

$$= 2X^2 + 2X + 3 \quad (\text{係数を} \bmod 5 \text{で計算})$$

$$(\text{積}) \quad A(X) * B(X) \bmod \Phi(X)$$

$$= (4X^2 + 3X + 2) * (3X^2 + 4X + 1) \bmod (X^2 + X + 1)$$

$$= 12X^4 + 25X^3 + 22X^2 + 11X + 2 \bmod (X^2 + X + 1)$$

$$= 2X^4 + 2X^2 + X + 2 \bmod (X^2 + X + 1) \quad (\text{係数を} \bmod 5 \text{で計算})$$

$$= X$$

として計算することができる。この事実を踏まえれば、実施例を構成することは容易である。

【0074】

素数 $p \geq 3$ の場合を考える。このとき、全ての楕円曲線は、 $E: y^2 = x^3 + ax + b$ の形に変換できる。これをWeierstrassの標準形と呼ぶ。ここで、 E 上の点の座標の各成分は、 $GF(p^n)$ の元であるので、 $A(X)$, $B(X)$ を $GF(p)[X]/(\Phi(X))$ の元として、 $(A(X), B(X))$ のように表現することができる。

【0075】

以下、 $P = (A(X), B(X))$ の2倍計算についての実施例を示す。和の計算は、 GF

(p)の場合に対する図 1 6 の実施例及び2倍計算の実施例から、容易に類推できるので、2倍計算の場合のみ説明する。

【 0 0 7 6 】

図 1 7 に2倍計算の実施例を示す。ステップ1601において、 $D1(X) = (2*B(X))^{(-1) \bmod \Phi(X)}$ を計算する。次に、ステップ1602にて、 $D2(X) = (3*A(X)^{2+a}) \bmod \Phi(X)$ を計算する。これらD1, D2に対して、テーブルD1[0](X) = D1(X), D2[0] = D2(X), $D1[1](X) = \Phi(X) - D1(X)$, $D2[1](X) = \Phi(X) - D2(X)$ を計算してRAMに格納する(ステップ1603)。D1[1](X), D2[1](X)は、それぞれ法 $\Phi(X)$ に対するD1(X), D2(X)の反転値である。次に2つ(2ビット)の乱数v, w(v, wは、共に0又は1)を発生する(ステップ1604)。次に、ステップ1603で求めたテーブルから、D1[v](X), D2[v](X)を取り出して、乗算剰余計算： $h(X) = D1[v](X)*D2[v](X) \bmod \Phi(X)$ を実行して結果をRAMに格納する(ステップ1605)。ここで、 $D1[1](X)*D2[1](X) \bmod \Phi(X) = (\Phi(X) - D1(X))*(\Phi(X) - D2(X)) \bmod \Phi(X) = D1(X)*D2(X) \bmod \Phi(X)$ であるので、ステップ1605の終了時には、v = 0, 1いずれの場合でも、正しいh(X)が求まる。次にステップ1606, 1607にて、A3(X)を求め、答えをRAMに格納する。ステップ1608にて、 $D3(X) = (A(X) - A3(X)) \bmod \Phi(X)$ を計算する。次に、このD3(X)と先に用いたh(X)に対して、テーブルh[0](X) = h(X), D3[0](X) = D3(X), $h[1](X) = \Phi(X) - h(X)$, $D3[1](X) = \Phi(X) - D3(X)$ を求めてRAMに格納する(ステップ1609)。h[1](X), D3[1](X)は、それぞれ、法 $\Phi(X)$ に対するh(X), D3(X)の反転値である。次に、ステップ1609で求めたテーブルから、h[w](X), D3[w](X)を取り出して、乗算剰余計算： $h[w](X)*D3[w](X) \bmod \Phi(X)$ を実行して結果をRAMに格納する(ステップ1610)。ここで、 $h[1](X)*D3[1](X) \bmod \Phi(X) = (\Phi(X) - h(X))*(\Phi(X) - D3(X)) \bmod \Phi(X) = h(X)*D3(X) \bmod \Phi(X)$ であるので、ステップ1610の終了時には、w = 0, 1いずれの場合でも、正しい値が求まる。最後に、ステップ1611にて、 $B3(X) = (B3(X) - B(X)) \bmod \Phi(X)$ を求めてRAMに格納する。これで、2Pの各座標の値が求められたことになる。これは、本発明の実施例の一つである。

【 0 0 7 7 】

本実施例にて、ステップ1601における逆数計算を実行する代表的な方法は、拡張ユークリッド互除法を利用するものとフェルマーの小定理を利用するものである。拡張ユークリッド互除法では、不定方程式： $2*B(X)*D1(X) + \Phi(X)*U(X) = 1$ を $(2*B(X))$ と $\Phi(X)$ の最大公約数を求める操作（ユークリッド互除法）を繰り返すことで、結果的に $D1(X)$ を求める方法である。一方、フェルマーの小定理を用いる方法とは、フェルマーの小定理、すなわち、 $\Phi(X)$ と互いに素な $g(X)$ に対し、 $g(X)^{(p^n-1) \bmod \Phi(X)} = 1$ が成り立つことから、 $g(X)^{(-1) \bmod \Phi(X)} = g(X)^{(p^n-2) \bmod \Phi(X)}$ となることを用いるものである。つまり、 $D1(X) = (2*B(X))^{(p^n-2) \bmod \Phi(X)}$ として、 $D1(X)$ を求めるものである。フェルマーの小定理を用いる方法では、逆数の計算は、べき乗剰余計算に帰着する。このべき乗剰余計算に本発明の方法を適用することは容易なことであり、オーバーフロー処理からのリーク情報を減らすことができる。

【0078】

次に、 $p = 2$ の場合の実施例を説明する。この場合は、これまでとは若干事情が異なる。これは、ガロア体 $GF(2^n)$ をreduction polynomial $\Phi(X)$ を用いて $GF(2)[X]/(\Phi(X))$ の形で表現した際、多項式の係数が $GF(2)$ であることから生ずるものである。 $GF(2)$ においては、符号は無意味である。というのは、この体の上では、 $-1 = 1$ だからである。このことから本発明の趣旨の一つである、「法 $\Phi(X)$ での反転」が無意味となるからである。また、 $GF(2)[X]$ の多項式 $A(X)$ の偶数倍も係数が $\bmod 2$ で計算されるため0となって、計算結果に影響を与えない。従って、 $-A(X) \equiv A(X)$ となる。よって、 $\Phi(X)$ と $A(X)$ の $GF(2)$ における線形結合は、0を除くと、本質的には、 $\Phi(X)$ 、 $\Phi(X)+A(X)$ 、 $A(X)$ だけとなる。この事実を踏まえて、 $p = 2$ の場合の実施例を示す。

【0079】

ここでは、Weierstrass標準形 $E: y^2 + xy = x^3 + ax + b$ を持つ楕円曲線を考える。 E 上の点 $P = (A(X), B(X))$ に対し、 $2P = (A3(X), B3(X))$ とする。実施例を構成するにあたって、2倍公式を次のように変形しておく。

【0080】

$$A3(X) = (h(X)^2 + h(X) + a) \bmod \Phi(X)$$

$$B3(X) = (h(X) * (A(X) + A3(X)) + A3(X) + B(X)) \bmod \Phi(X)$$

ここで、 $h(X)$ は、 $h(X) = (A(X)^2 + B(X)) * A(X)^{-1} \bmod \Phi(X)$ とする。

【 0 0 8 1 】

図 1 8 に 2 倍計算の実施例を示す。ステップ 1701 において、 $D1(X) = (A(X))^{-1} \bmod \Phi(X)$ を計算する。次に、ステップ 1702 にて、 $D2(X) = (A(X)^2 + B(X)) \bmod \Phi(X)$ を計算する。これら $D1(X)$ 、 $D2(X)$ に対して、テーブル $D1[0](X) = D1(X)$ 、 $D2[0](X) = D2(X)$ 、 $D1[1](X) = \Phi(X) + D1(X)$ 、 $D2[1](X) = \Phi(X) + D2(X)$ を計算して RAM に格納する (ステップ 1703)。次に 4 つ (4 ビット) の乱数 v 、 w 、 i 、 j (v 、 w 、 i 、 j は、全て 0 又は 1) を発生する (ステップ 1704)。次に、ステップ 1703 で求めたテーブルから、 $D1[v](X)$ 、 $D2[w](X)$ を取り出して、乗算剰余計算： $h(X) = D1[v](X) * D2[w](X) \bmod \Phi(X)$ を実行して結果を RAM に格納する (ステップ 1705)。ここで $(\Phi(X) + D1(X)) * D2(X) \bmod \Phi(X) = D1(X) * (\Phi(X) + D2(X)) \bmod \Phi(X) = (\Phi(X) + D1(X)) * (\Phi(X) + D2(X)) \bmod \Phi(X) = D1(X) * D2(X) \bmod \Phi(X)$ であるので、ステップ 1705 の終了時には、 v 、 $w = 0$ 、1 のどの場合でも、正しい $h(X)$ が求まる。次にステップ 1706、1707 にて、 $A3(X)$ を求め、答えを RAM に格納する。ステップ 1708 にて、 $D3(X) = (A(X) + A3(X)) \bmod \Phi(X)$ を計算する。次に、この $D3(X)$ と先に用いた $h(X)$ に対して、テーブル $h[0](X) = h(X)$ 、 $D3[0](X) = D3(X)$ 、 $h[1](X) = \Phi(X) + h(X)$ 、 $D3[1](X) = \Phi(X) + D3(X)$ を求めて RAM に格納する (ステップ 1709)。次に、ステップ 1709 で求めたテーブルから、 $h[i](X)$ 、 $D3[j](X)$ を取り出して、乗算剰余計算： $h[i](X) * D3[j](X) \bmod \Phi(X)$ を実行して結果を RAM に格納する (ステップ 1710)。ここで、 $(\Phi(X) + h(X)) * D3(X) \bmod \Phi(X) = h(X) * (\Phi(X) + D3(X)) \bmod \Phi(X) = (\Phi(X) + h(X)) * (\Phi(X) + D3(X)) \bmod \Phi(X) = h(X) * D3(X) \bmod \Phi(X)$ であるので、ステップ 1710 の終了時には、 i 、 $j = 0$ 、1 のどの場合でも、正しい値が求まる。最後に、ステップ 1711 にて、 $B3(X) = (B3(X) + A3(X) + B(X)) \bmod \Phi(X)$ を求めて RAM に格納する。これで、2P の各座標の値が求められたことになる。これは、本発明の実施例の一つである。

【 0 0 8 2 】

本実施例にて、ステップ 1701 における逆数計算を実行する代表的な方法は、拡

張ユークリッド互除法を利用するものとフェルマーの小定理を利用するものである。拡張ユークリッド互除法では、不定方程式： $A(X) \cdot D1(X) + \Phi(X) \cdot U(X) = 1$ を $A(X)$ と $\Phi(X)$ の最大公約数を求める操作（ユークリッド互除法）を繰り返すことで、結果的に $D1(X)$ を求める方法である。一方、フェルマーの小定理を用いる方法とは、フェルマーの小定理、すなわち、 $\Phi(X)$ と互いに素な $g(X)$ に対し、 $g(X)^{(2^n-1) \bmod \Phi(X)} = 1$ が成り立つことから、 $g(X)^{(-1) \bmod \Phi(X)} = g(X)^{(2^n-2) \bmod \Phi(X)}$ となることを用いるものである。つまり、 $D1(X) = (A(X))^{(2^n-2) \bmod \Phi(X)}$ として、 $D1(X)$ を求めるものである。フェルマーの小定理を用いる方法では、逆数の計算は、べき乗剰余計算に帰着する。このべき乗剰余計算に本発明の方法を適用することは容易なことであり、オーバーフロー処理からのリーク情報を減らすことができる。

【 0 0 8 3 】

楕円曲線上の点のモデル・ヴェイユ群演算に関する上記実施例は、いずれも、符号の影響が、即時に解消されるものであった。ここで、即時解消されないような場合の実施例を示す。

【 0 0 8 4 】

図 1 9 は、 $GF(p)$ 上の楕円曲線 $E: y^2 = x^3 + ax + b$ 上の点 $P = (x, y)$ の2倍計算に本発明を適用したものである。以下、 $2P = (x_3, y_3)$ とする。ステップ1801において、 $D1 = (2 \cdot y)^{(-1) \bmod p}$ を計算する。次に、ステップ1802にて、 $D2 = (3 \cdot x^2 + a) \bmod p$ を計算する。 $D2$ に対して、テーブル $D2[0] = D2$, $D2[1] = p - D2$ を計算してRAMに格納する（ステップ1803）。 $D2[1]$ は、法 p に対する $D2$ の反転値である。次に2つ（2ビット）の乱数 v, w （ v, w は、共に0又は1）を発生する（ステップ1804）。次に、ステップ1803で求めたテーブルから、 $D2[v]$ を取り出して、乗算剰余計算： $h = D1 \cdot D2[v] \bmod p$ を実行して結果をRAMに格納する（ステップ1805）。ここで、 $D1 \cdot D2[1] \bmod p$ は、 $D1 \cdot D2 \bmod p$ であるか、または、 $p - (D1 \cdot D2 \bmod p)$ であり、図 1 5 の実施例とは異なり、ステップ1805終了時点では、正しい値になるとは限らない。次にステップ1806, 1807にて、 x_3 を求め、答えをRAMに格納する。ここで、ステップ1806において、 h が $\bmod p$ で2乗されるため、ステップ1805の結果が、 $D1 \cdot D2 \bmod p$ であっても、 $p - (D1 \cdot D2 \bmod p)$

p)であっても、正しい x_3 の値が求まる。ステップ1808にて、 $D_3 = (x - x_3) \bmod p$ を計算する。次に、この D_3 に対して、テーブル $D_3[0] = D_3$, $D_3[1] = p - D_3$ を求めてRAMに格納する（ステップ1809）。 $D_3[1]$ は、法 p に対する D_3 の反転値である。次に、ステップ1809で求めたテーブルから、 $D_3[w]$ を取り出して、乗算剰余計算： $h * D_3[w] \bmod p$ を実行して結果をRAMに格納する（ステップ1810）。ここで、 $h * D_3[1] \bmod p = h * (p - D_3) \bmod p = p - (h * D_3 \bmod p)$ であるので、ステップ1810の終了時では、 $v = 0$ であれば、 $h * D_3 \bmod p$ 、 $v = 1$ であれば、 $p - (h * D_3 \bmod p)$ となっている。ステップ1811にて、テーブル $y_3[0] = y_3$, $y_3[1] = p - y_3$ を求めてRAMに格納する。次に、条件分岐処理（ステップ1812）にて、 v と w の排他的論理和(exclusive OR) $v \text{ EXOR } w$ の値が、0であるか1であるかを判定し、0であれば、ステップ1813に進み、1であれば、ステップ1814に進む。排他的論理和 $v \text{ EXOR } w$ は、 v と w が同じであれば0、異なれば1となる。従って、最終的に、該排他的論理和が0のときには、ステップ1813によって正しい y_3 が求められ、1のときは、ステップ1814にて、正しい y_3 が求められる。これで、 $2P$ の各座標の値が求められたことになる。これは、本発明の実施例の一つである。

【0085】

本実施例にて、ステップ1801における逆数計算を実行する代表的な方法は、拡張ユークリッド互除法を利用するものとフェルマーの小定理を利用するものである。拡張ユークリッド互除法では、不定方程式： $2 * y * D_1 + p * U = 1$ を($2 * y$ と p の最大公約数を求める操作（ユークリッド互除法）を繰り返すことで、結果的に D_1 を求める方法である。一方、フェルマーの小定理を用いる方法とは、フェルマーの小定理、すなわち、 p と互いに素な正の整数 g に対し、 $g^{p-1} \bmod p = 1$ が成り立つことから、 $g^{p-1} \bmod p = g^{p-2} \bmod p$ となることを用いるものである。つまり、 $D_1 = (2 * y)^{p-2} \bmod p$ として、 D_1 を求めるものである。フェルマーの小定理を用いる方法では、逆数の計算は、べき乗剰余計算に帰着する。このべき乗剰余計算に本発明の方法を適用することは容易なことであり、オーバーフロー処理からのリーク情報を減らすことができる。

【0086】

また、本実施例を、図16の実施例等に倣って $GF(p^n)$ の上の楕円曲線上の2倍

算に拡張することは、全く容易である。

【 0 0 8 7 】

【発明の効果】

本発明によれば、ＩＣカードチップなどの情報処理装置において、データとして本来の値とは異なる「モジュラ反転値」を用いて剰余乗算の計算を行い、その計算結果を修正することで正しい結果を求めることにより、消費電流の波形から処理や暗号鍵の推測を行うことが困難になる。

【図面の簡単な説明】

【図 1】

ＩＣカードの概観及び端子の構成を示す図である。

【図 2】

マイクロコンピュータの構成図である。

【図 3】

消費電流の波形の例を示す図である。

【図 4】

べき乗剰余計算（ウィンドウ幅２ビットのアディション・チェイン方式）の処理手順を示す図である。

【図 5】

べき乗剰余計算（ウィンドウ幅２ビットのスライディング・ウィンドウ方式）の処理手順を示す図である。

【図 6】

楕円曲線の形状を示す図である。

【図 7】

楕円曲線上の加法を説明する図である。

【図 8】

楕円曲線上の点 P のスカラー倍の計算アルゴリズム（２ビットのアディション・チェイン方式）を示す図である。

【図 9】

モンゴメリ法を用いたべき乗剰余計算に本発明を適用した場合の第 1 の実施例

の処理手順を示す図である。

【図 1 0】

モンゴメリ法を用いたべき乗剰余計算に本発明を適用した場合の第 2 の実施例の処理手順を示す図である。

【図 1 1】

通常のべき乗剰余計算に本発明を適用した場合の第 1 の実施例の処理手順を示す図である。

【図 1 2】

通常のべき乗剰余計算に本発明を適用した場合の第 2 の実施例の処理手順を示す図である。

【図 1 3】

モンゴメリ法を用いたべき乗剰余計算に本発明を適用した場合の第 3 の実施例の処理手順を示す図である。

【図 1 4】

モンゴメリ法を用いたべき乗剰余計算に本発明を適用した場合の第 3 の実施例の処理手順を示す図（続き）である。

【図 1 5】

$GF(p)$ 上の楕円曲線上の点の 2 倍計算に本発明を適用した場合の第 1 の実施例の処理手順を示す図である。

【図 1 6】

$GF(p)$ 上の楕円曲線上の点の和計算に本発明を適用した場合の第 1 の実施例の処理手順を示す図である。

【図 1 7】

$GF(p^n)$ 上の楕円曲線上の点の 2 倍計算に本発明を適用した場合の実施例の処理手順を示す図である。

【図 1 8】

$GF(2^n)$ 上の楕円曲線上の点の 2 倍計算に本発明を適用した場合の実施例の処理手順を示す図である。

【図 1 9】

GF(p)上の楕円曲線上の点の和計算に本発明を適用した場合の第2の実施例の処理手順を示す図である。

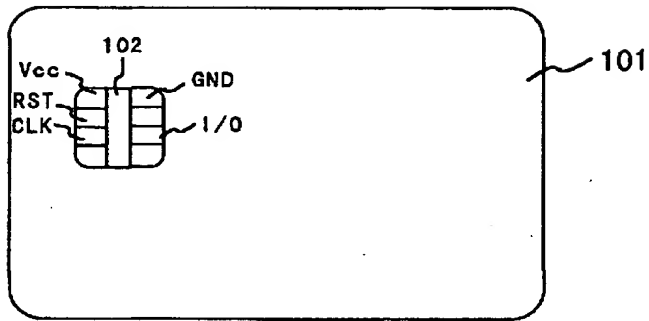
【符号の説明】

906,1005,1106,1207,1304,1404,1604…ステップ（剰余乗算の選択）、912～915,1012～1015,1112～1115,1213～1216,1311～1314,1405,1410,1605,1610…ステップ（選択的な剰余乗算）、917,1017,1116,1217,1317…（結果の補正）、919～920,1019～1020,1118～1119,1219～1220,1319～1320…ステップ（結果の出力）

【書類名】 図面

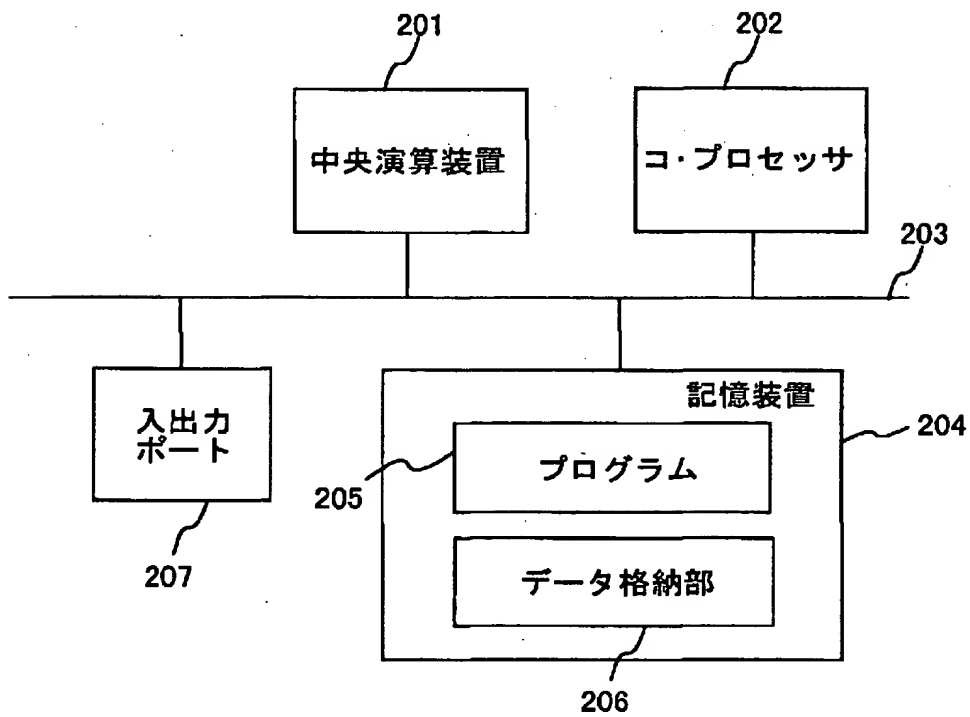
【図 1】

図 1



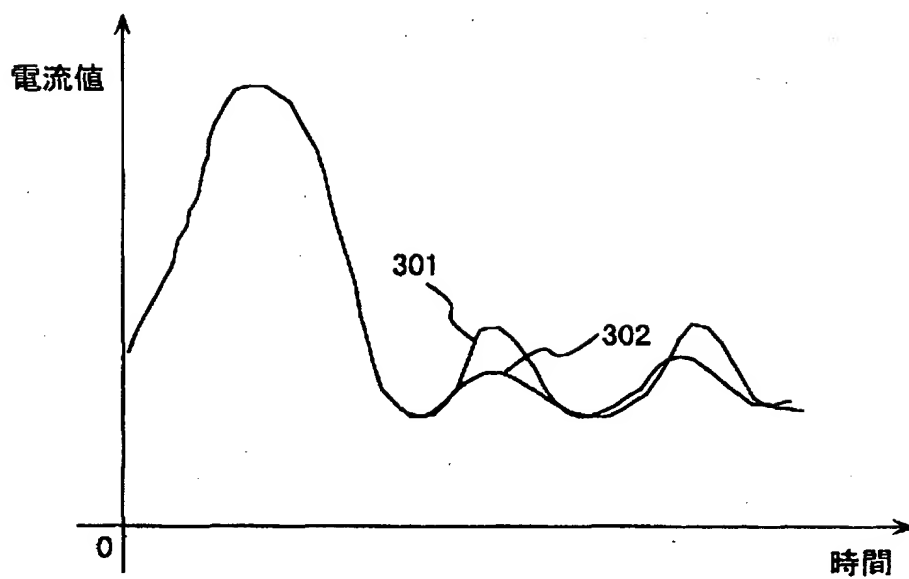
【図 2】

図 2



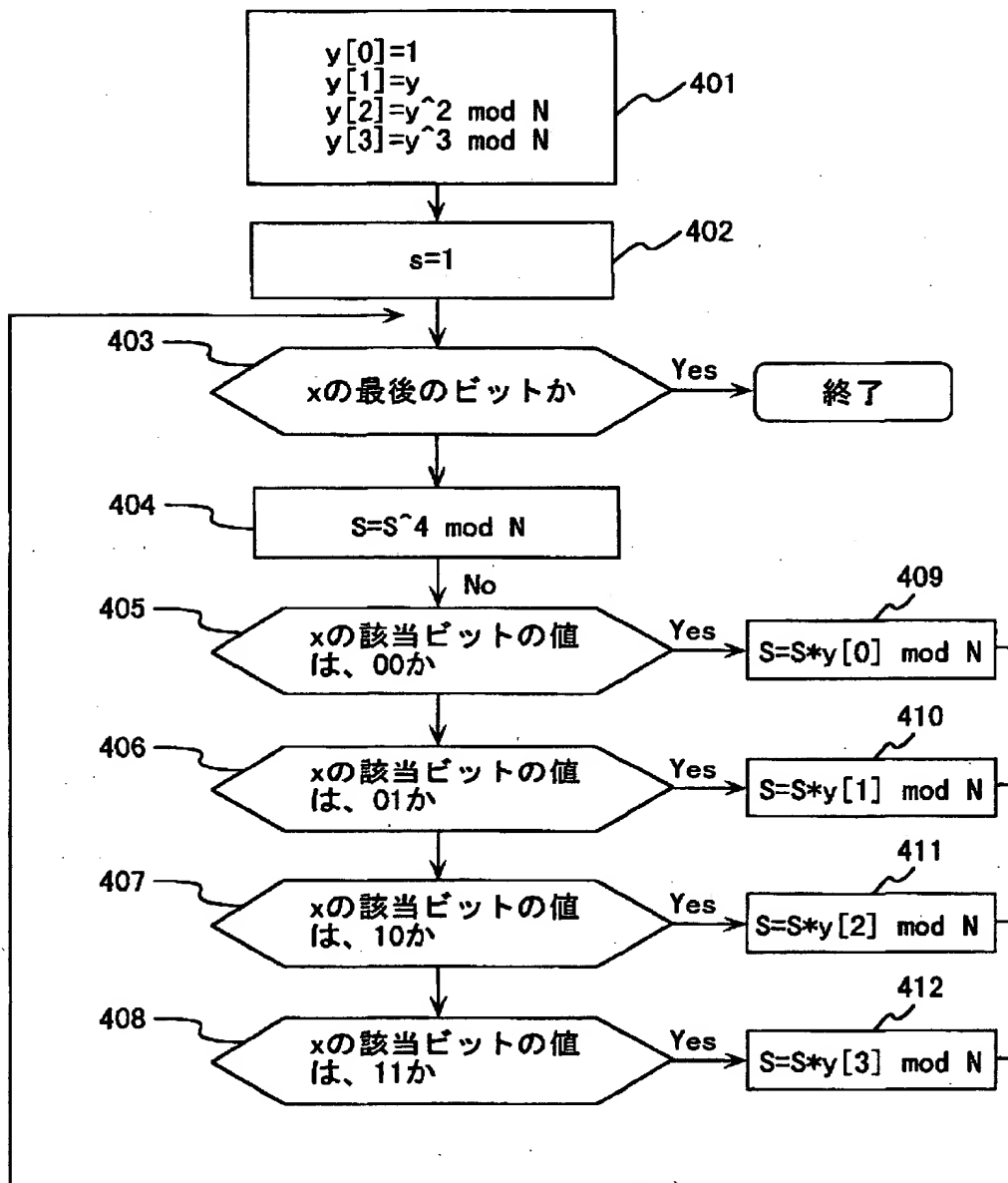
【図 3】

図 3



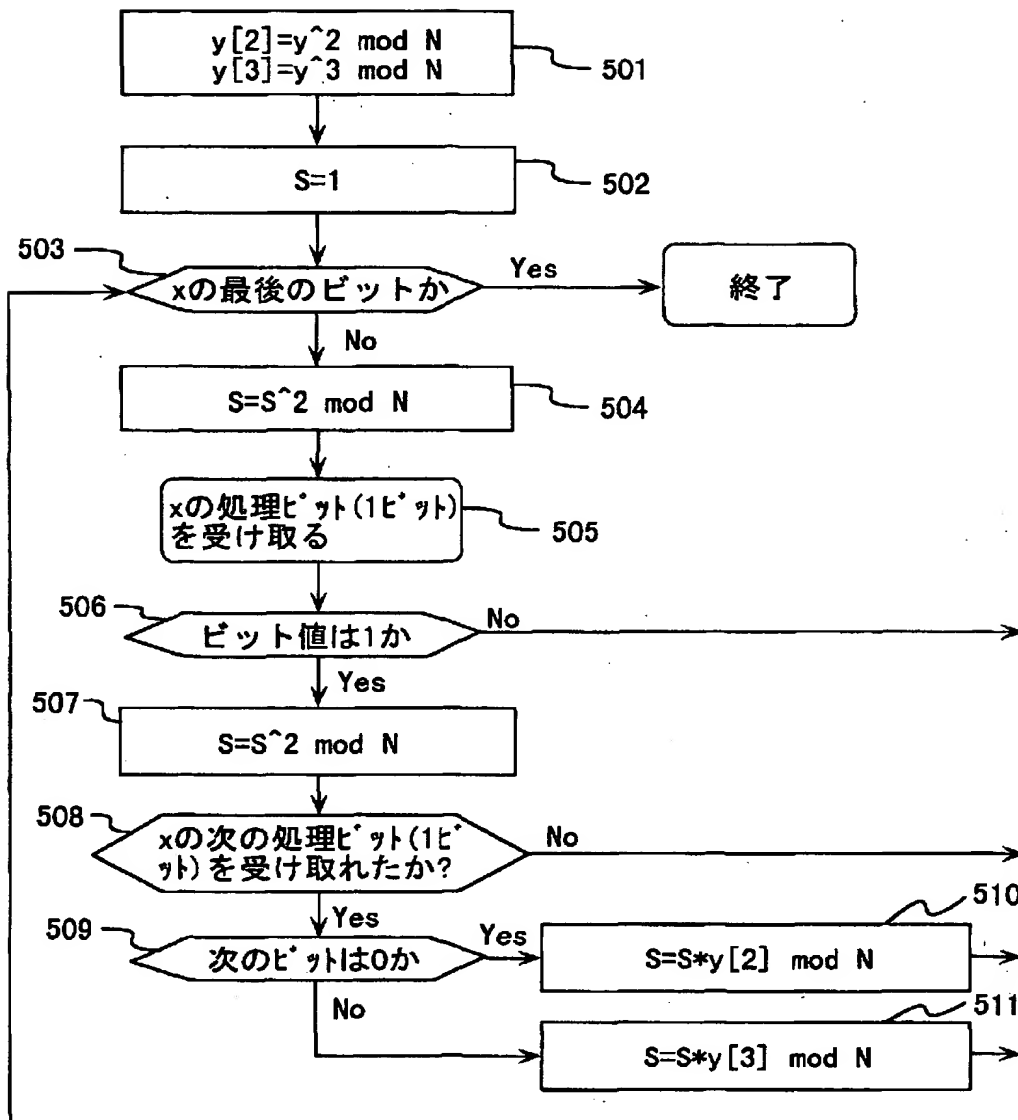
【図 4】

図 4



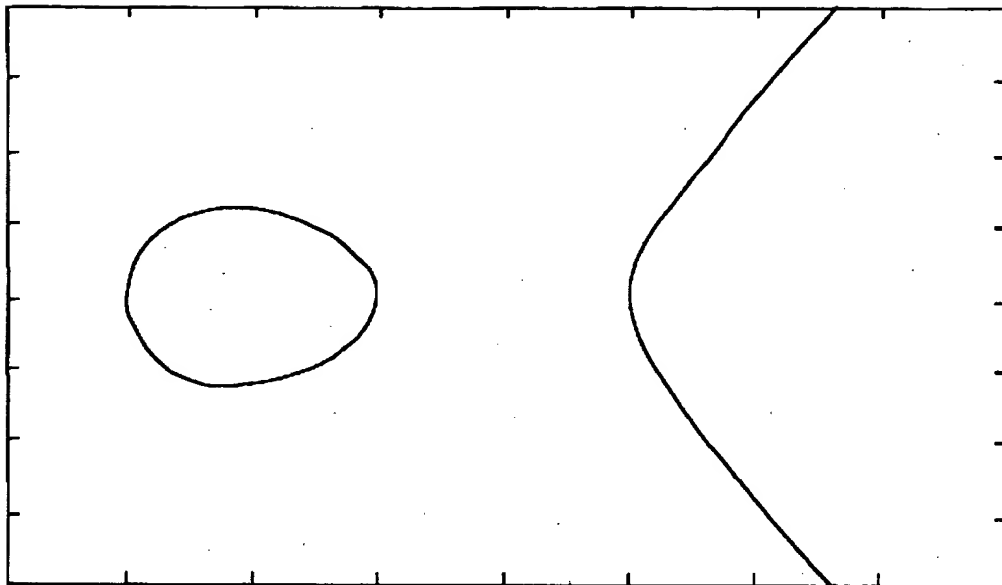
【図 5】

図 5



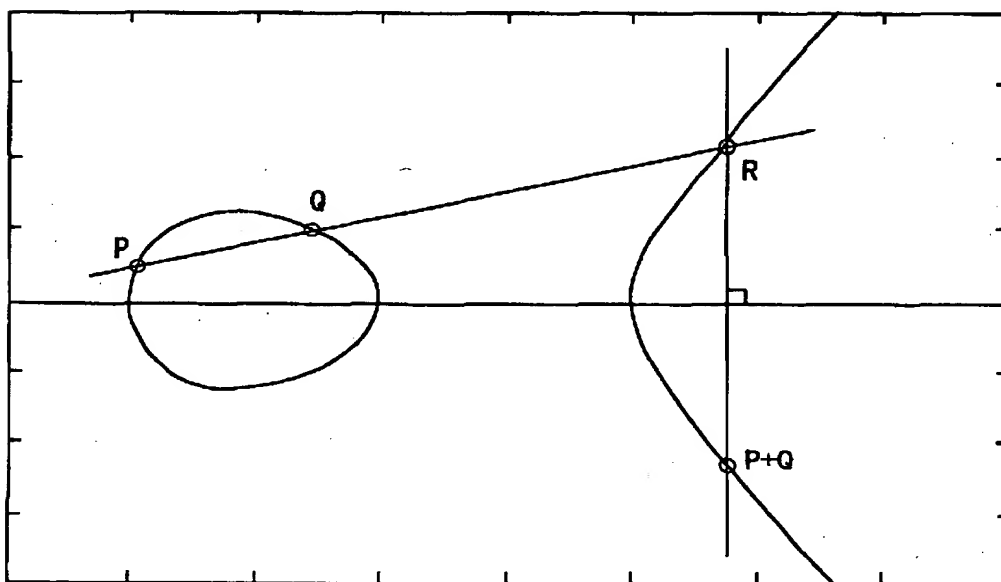
【図 6】

図 6



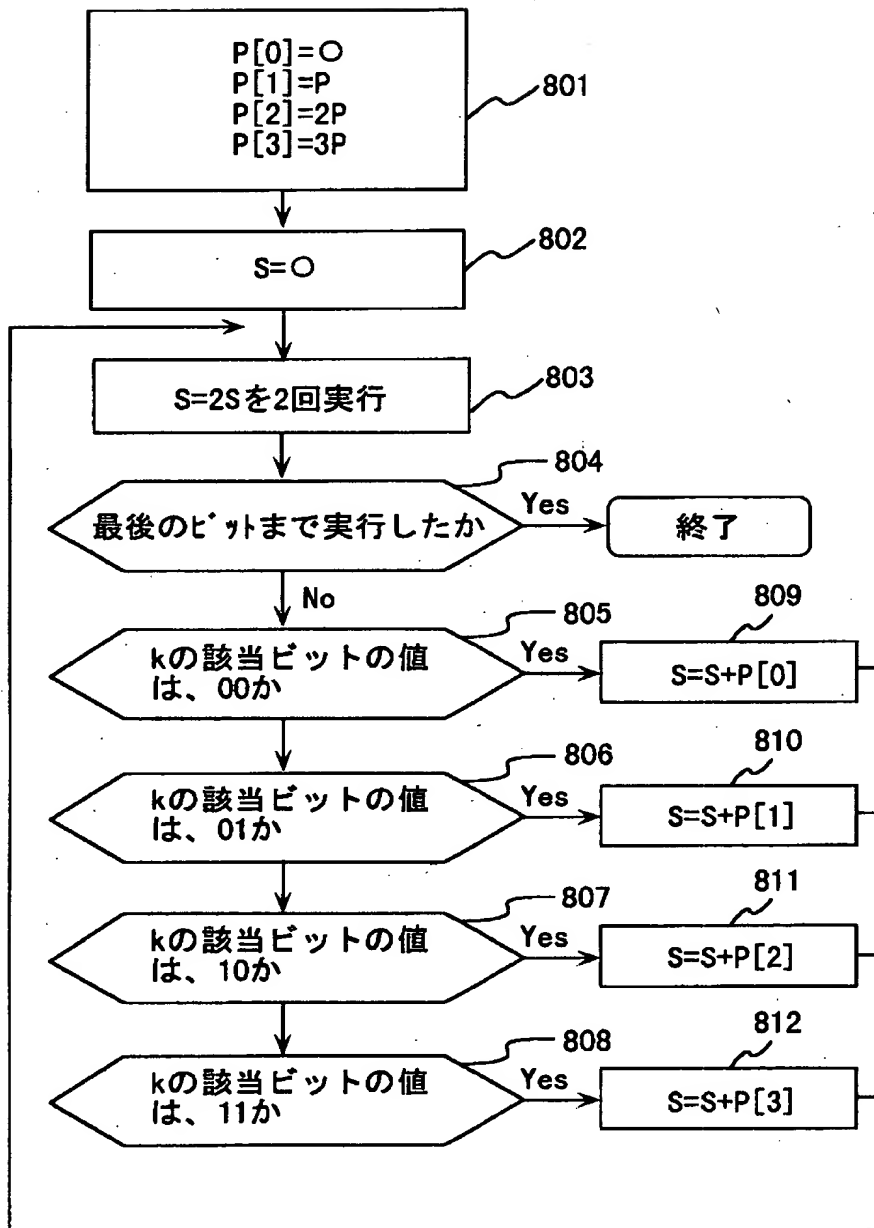
【図 7】

図 7



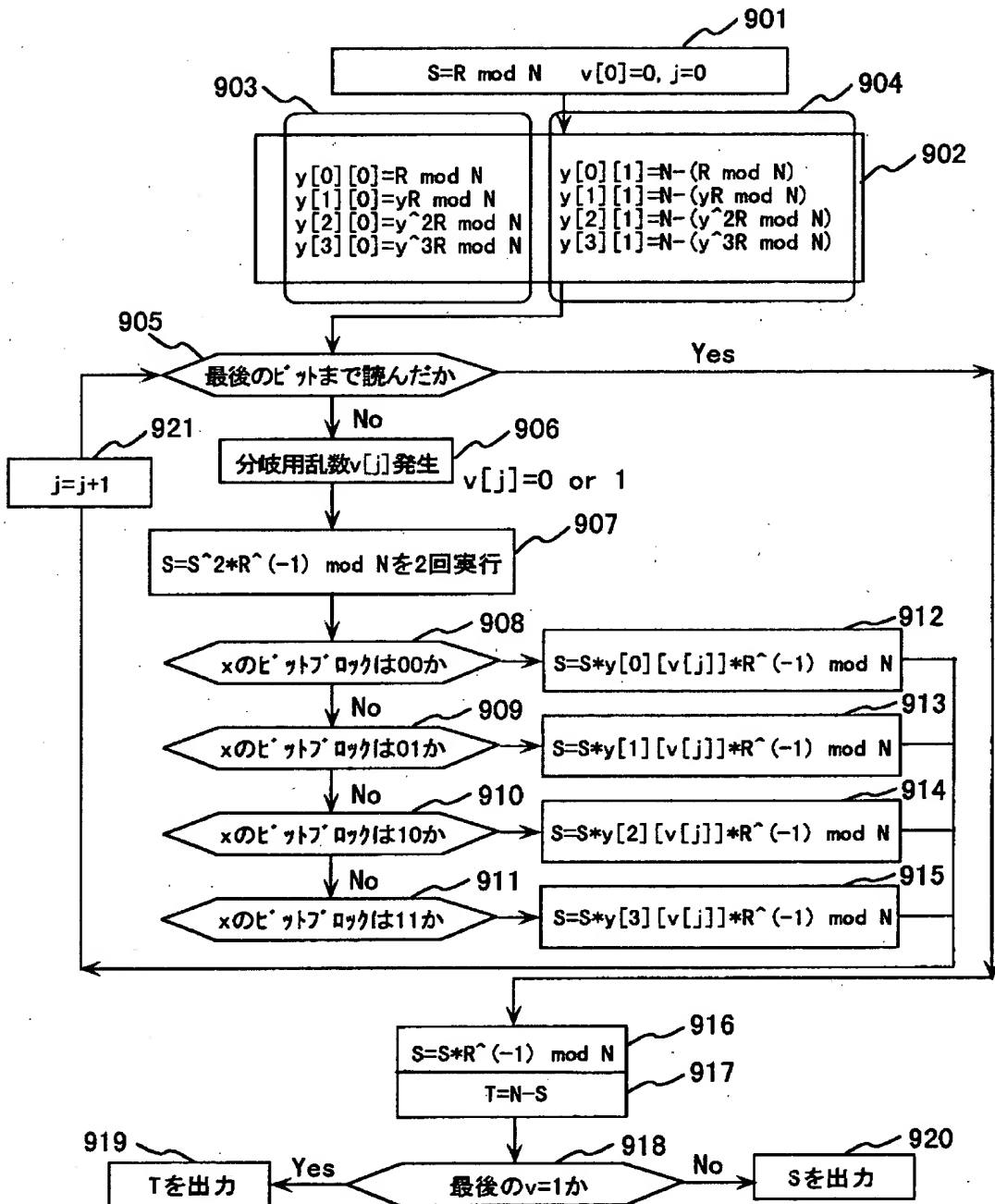
【図 8】

図 8



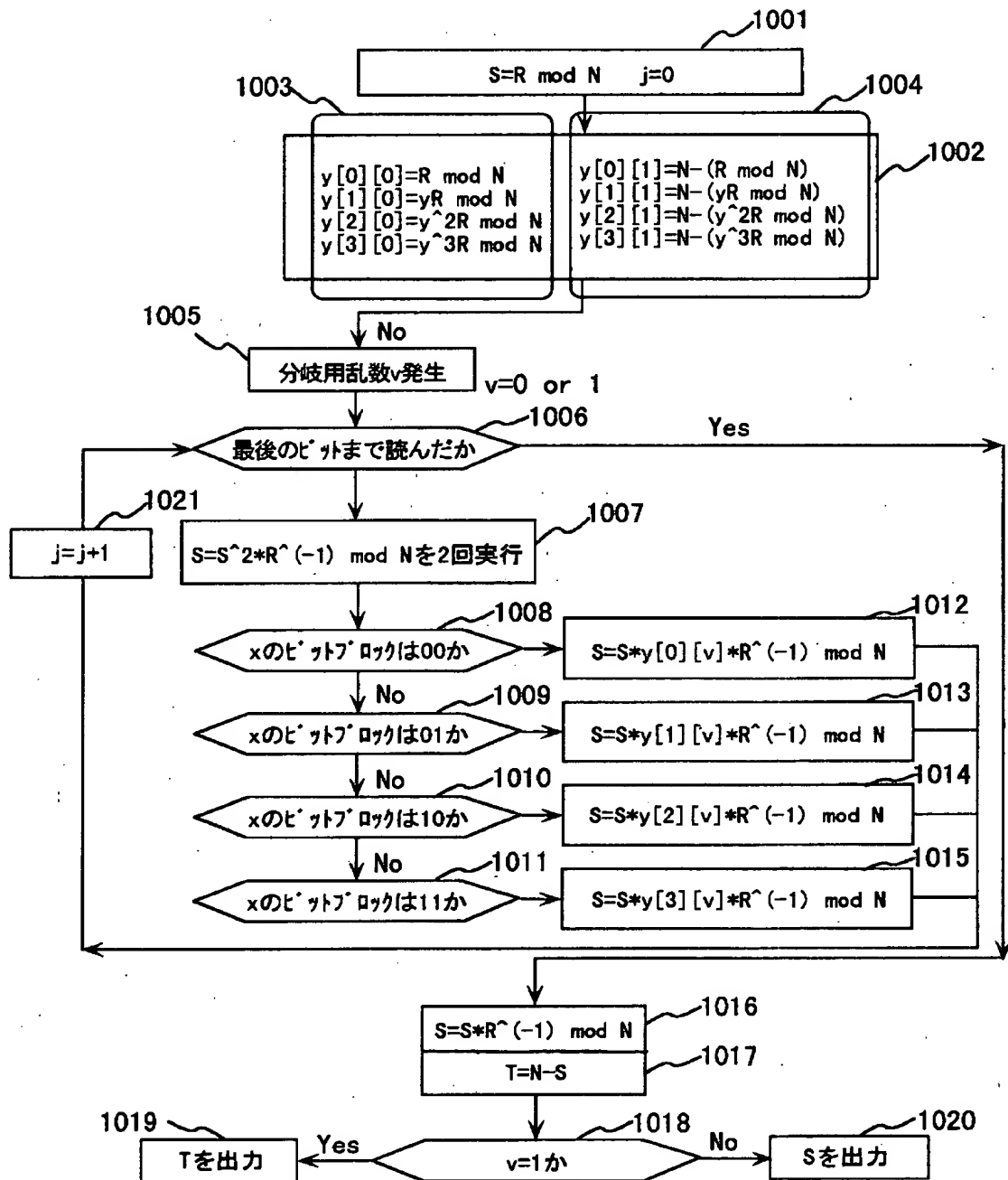
【図 9】

図 9



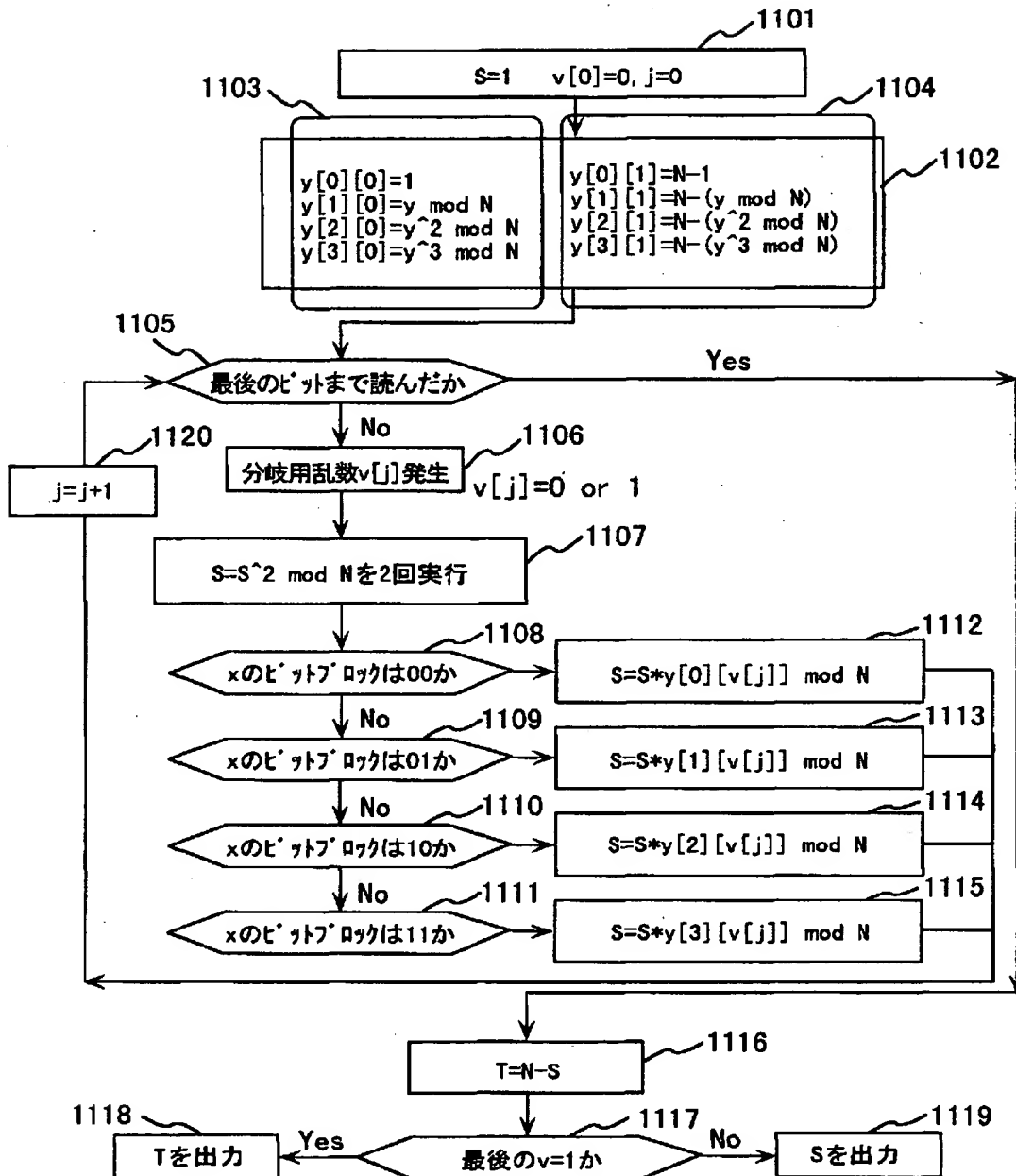
【図10】

図 10



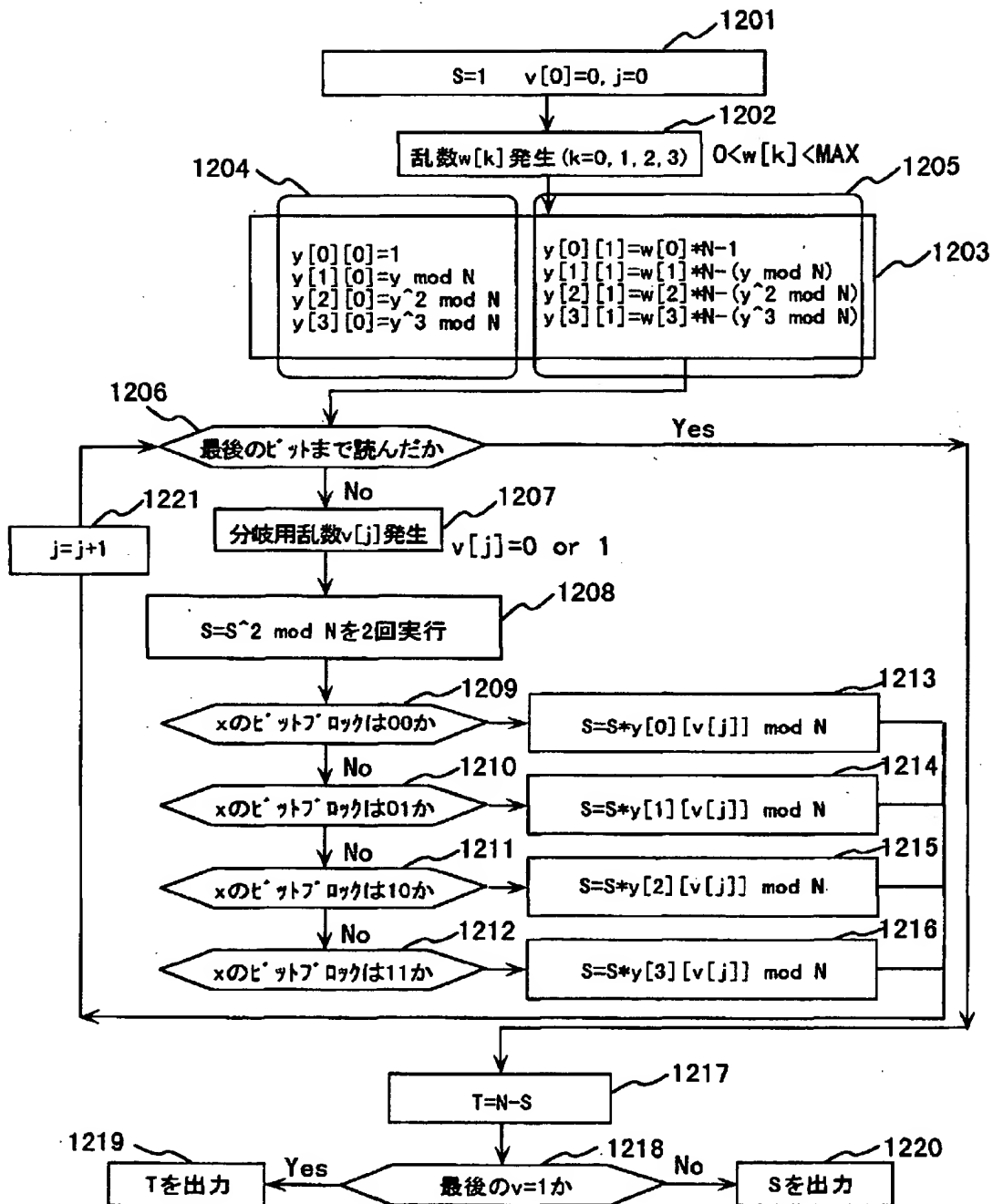
【図 11】

図 11



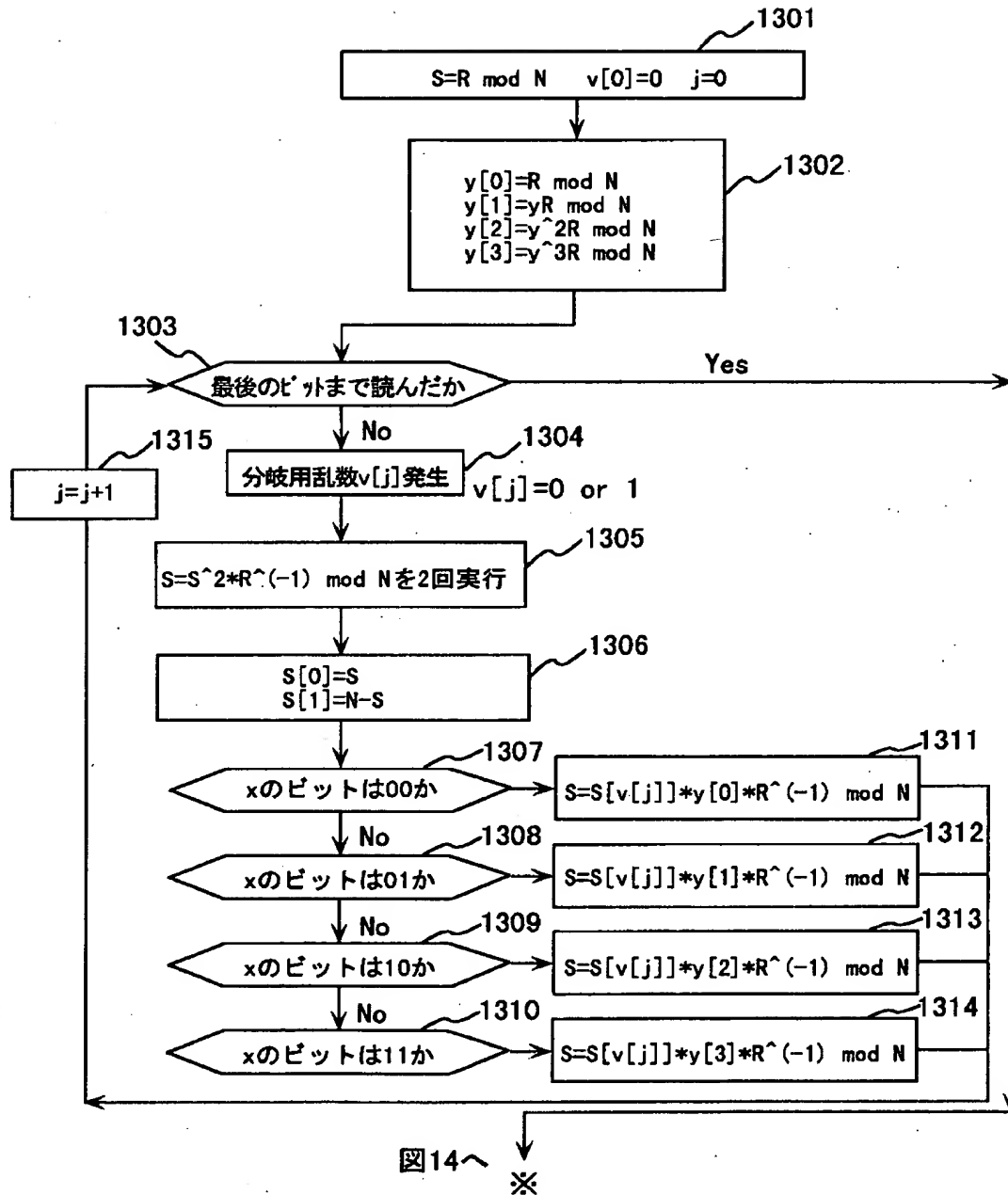
【図 1 2】

図 1 2



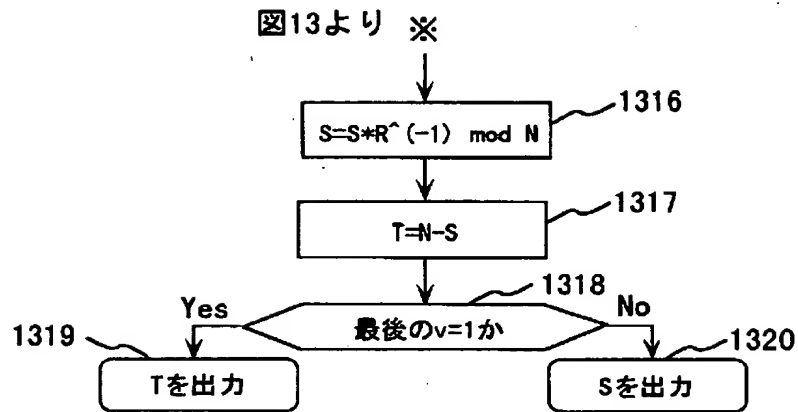
【図 13】

図 13



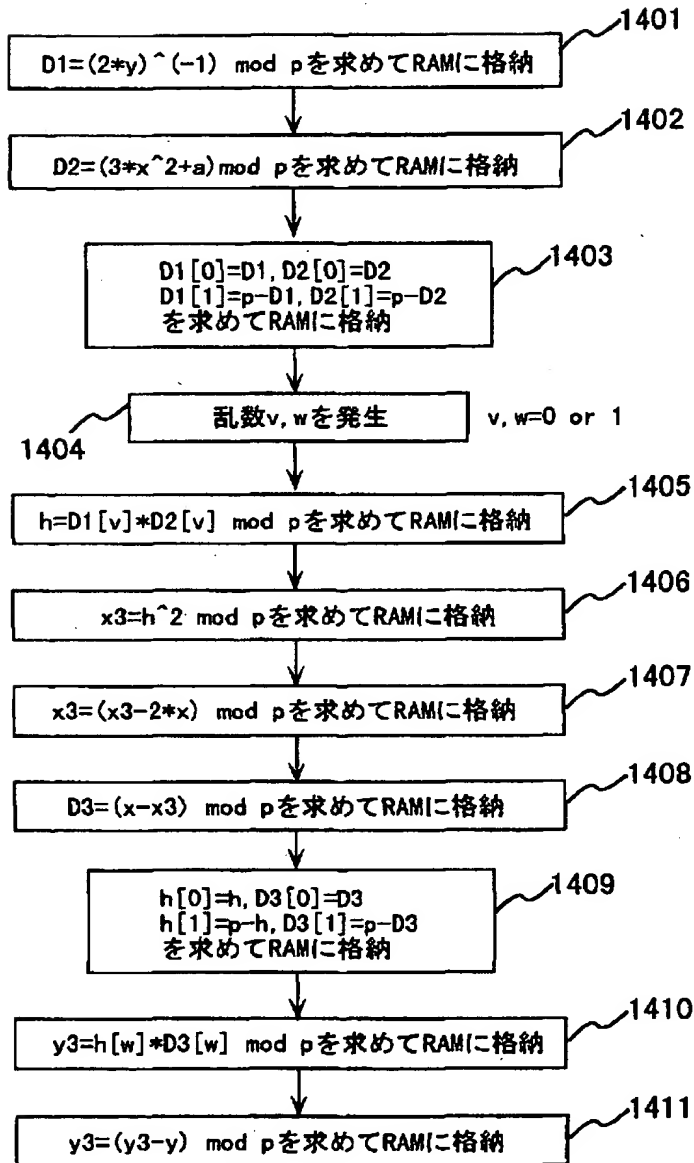
【図 14】

図 14



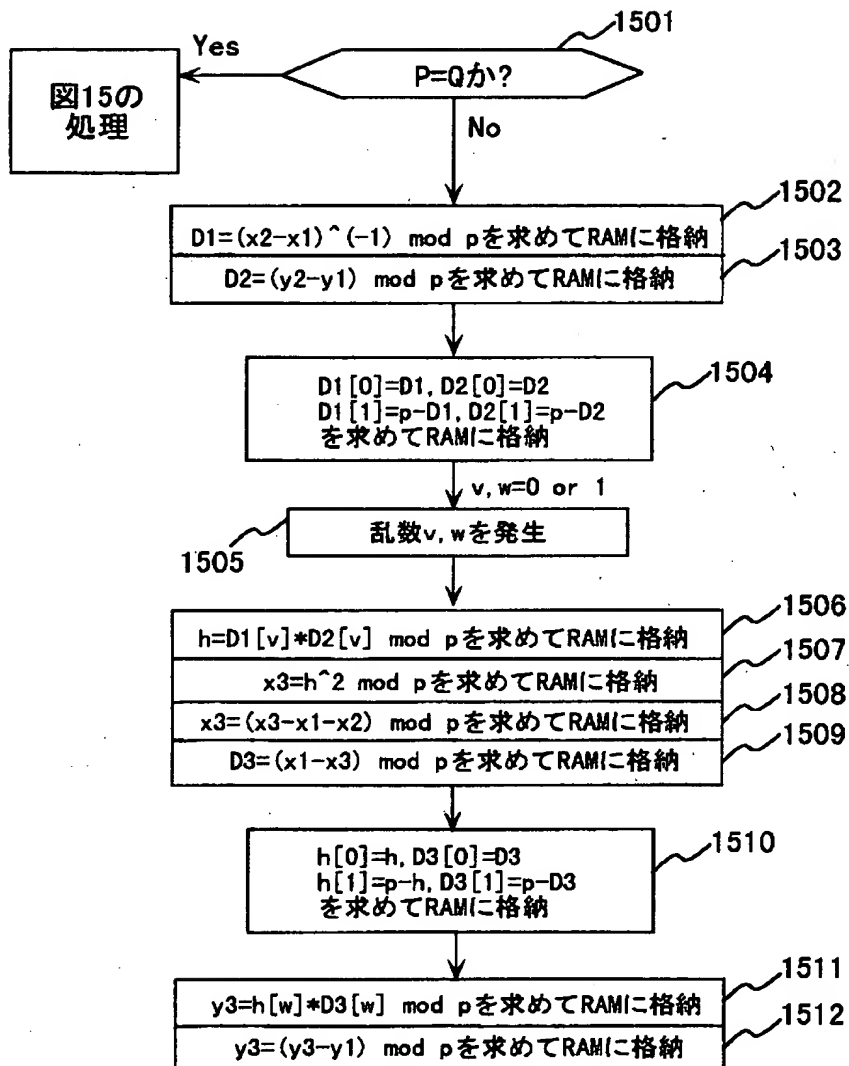
【図 15】

図 15



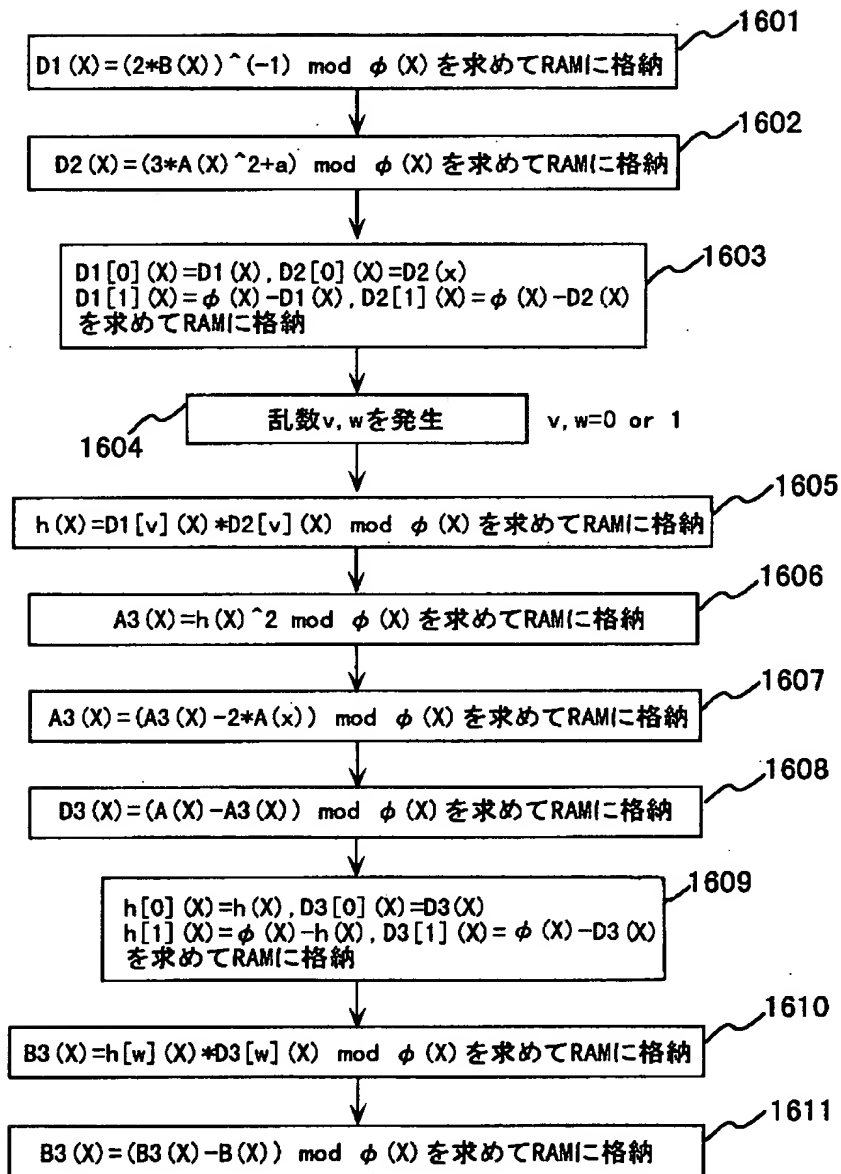
【図 16】

図 16



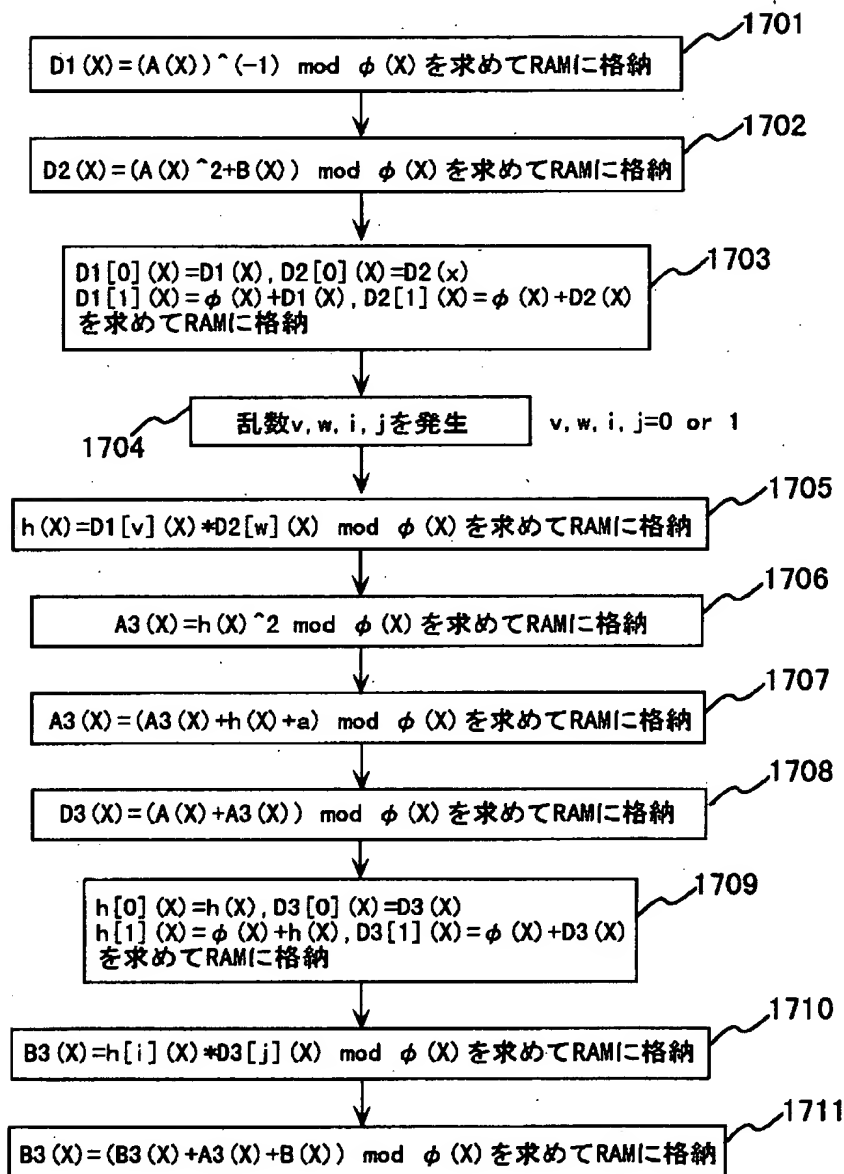
【図 17】

図 17



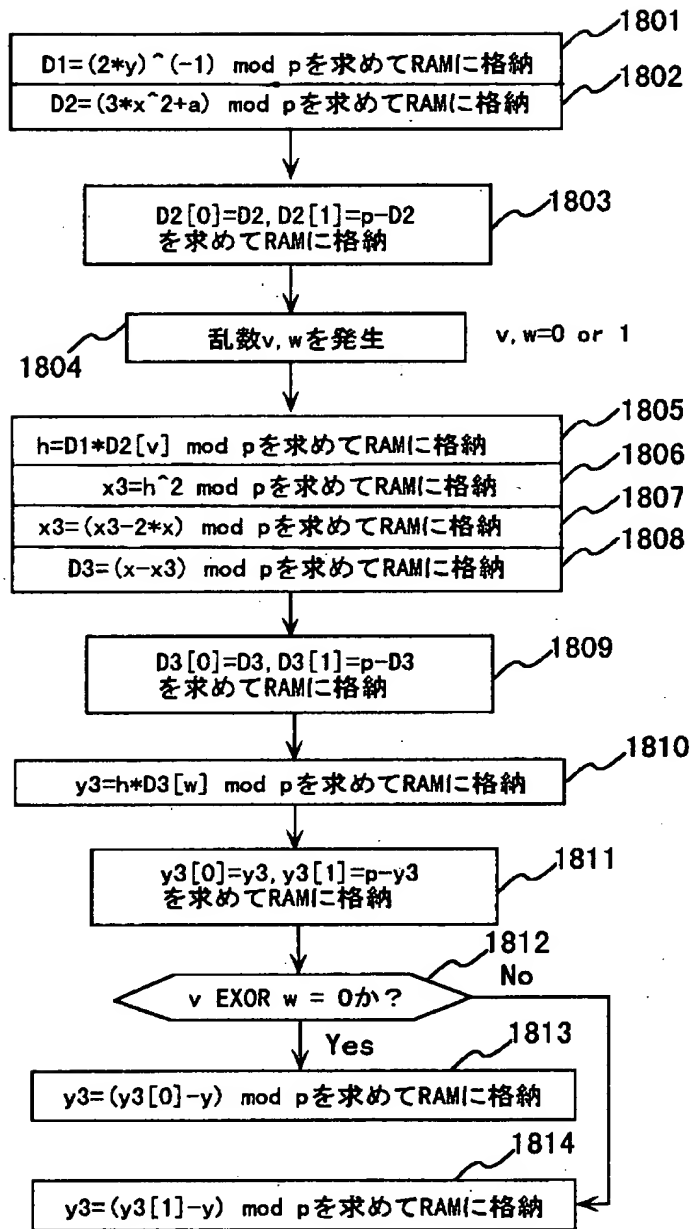
【図 18】

図 18



【図 19】

図 19



【書類名】 要約書

【要約】

【課題】 ICカードなどの情報処理装置において、暗号処理中に行われる剰余乗算演算の際に発生するオーバーフロー処理は、特有の消費電流を示す。この問題に対してデータ処理と消費電流との関連性を減らす。

【解決手段】 2ビットアディション・チェイン方式に従ってべき剰余計算を行う処理手順において、ステップ1106で実行する剰余乗算をランダムに選択し、ステップ1112～1115で選択された剰余乗算を実行し、ステップ1116で結果の補正を行い、ステップ1118または1119で計算結果（補正しない値または補正した値）を出力する。

【選択図】 図11

出 願 人 履 歴 情 報

識別番号 [000005108]

1. 変更年月日 1990年 8月31日

[変更理由] 新規登録

住 所 東京都千代田区神田駿河台4丁目6番地

氏 名 株式会社日立製作所